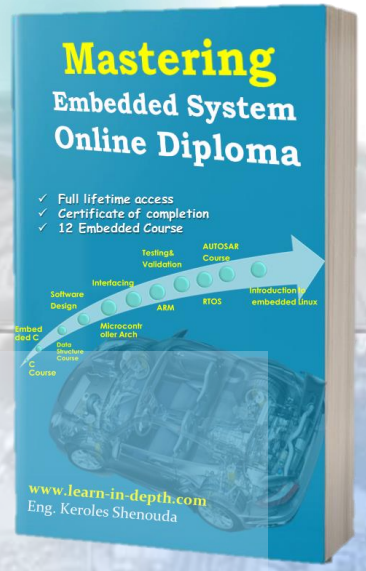
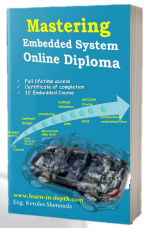
Mastering Embedded System

Online Diploma

- ✓ Full lifetime access
- ✓ Access on Android mobile and PC (Windows)
- ✓ Certificate of completion
- ✓ 12 Embedded Course



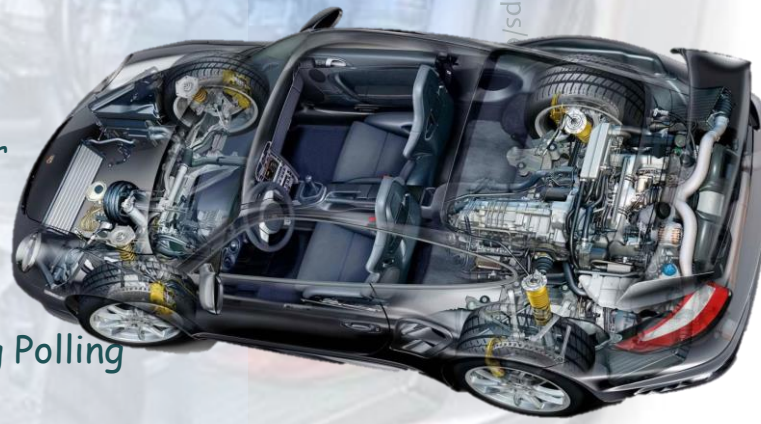


#LEARN_IN DEPTH

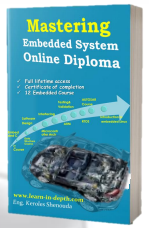
#Be_professional_in_embedded_system

Unit 8 (MCU Interfacing) . lesson 3 UART Controller

- **USART** Logic Level in STM32F103XX
- **NRZ (non-return-to-zero)**
- UART Over Sampling Mechanism
- send or receive data via UART ports by using three different methods
  - Polling Mechanism
  - Interrupt Mechanism
  - DMA Mechanism
- How to read USART Chapter from TRM for X SoC
  - Figure out **USART main features**
  - USART Functional Description
  - USART Block Diagram
  - Focused on UART Mode
  - Registers
  - TX Behavior
  - RX Behavior
  - Interrupt Behavior
- How to write UART Driver
  - Figure out Configurable parameters
  - Add USART registers on device header
  - Add Configuration Structure
  - BaudRate: Clock Code Calculations
  - Partial Implementation on RCC Driver
  - IRQ Handling Support
  - APIs Implementation
- CASE Study 1 with Debugging : using Polling mechanism
- CASE Study 2 with Debugging : using Interrupt mechanism

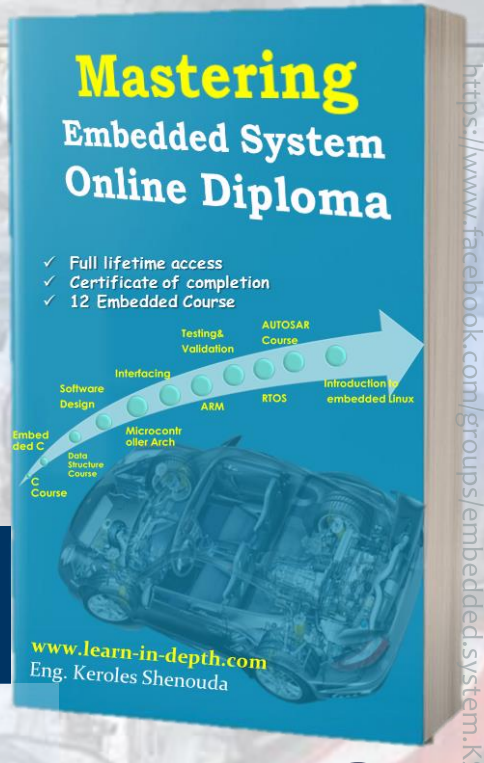


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



Logic Level : Stm32F103C8X

LOGIC LEVELS

LEARN-IN-DEPTH

Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Logic level For Input

Table 35. I/O static characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Low level input voltage	Standard IO input low level voltage	-	-	$0.28*(V_{DD}-2\ V)+0.8\ V^{(1)}$	V
		IO FT(3) input low level voltage	-	-	$0.32*(V_{DD}-2V)+0.75\ V^{(1)}$	
		All I/Os except BOOT0	-	-	$0.35V_{DD}^{(2)}$	
V_{IH}	High level input voltage	Standard IO input high level voltage	$0.41*(V_{DD}-2\ V)+1.3\ V^{(1)}$	-	-	
		IO FT(3) input high level voltage	$0.42*(V_{DD}-2\ V)+1\ V^{(1)}$	-	-	
		All I/Os except BOOT0	$0.65V_{DD}^{(2)}$	-	-	

Logic level

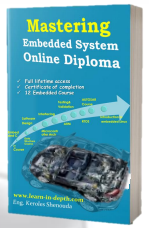
Table 35. I/O static characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Low level input voltage	Standard IO input low level voltage	-	-	0.28*(V_{DD}-2 V)+0.8 V(1)	V
		IO FT(3) input low level voltage	-	-	0.32*(V_{DD}-2V)+0.75 V(1)	
		All I/Os except BOOT0	-	-	0.35V_{DD}(2)	
V_{IH}	High level input voltage	Standard IO input high level voltage	0.41*(V_{DD}-2 V)+1.3 V(1)	-	-	
		IO FT(3) input high level voltage	0.42*(V_{DD}-2 V)+1 V(1)	-	-	
		All I/Os except BOOT0	0.65V_{DD}(2)	-	-	

FOLLOW US

Press here

4



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Five Volt tolerant pin

VI_Max = VDD + 4V

VI Steady Max = 5.5

VIH(Min) = 0.42*(Vdd-2)+1

VIL(Max) = 0.32*(Vdd-2)+0.75

VI Steady Min = -0.3

VI_Min = Vss-0.3 V

Three Volt tolerant pin

VI_Max = 4 V

VI Steady Max = VDD+0.3

VIH(Min) = 0.41*(Vdd-2)+1.3

VIL(Max) = 0.28*(Vdd-2)+0.8

VI Steady Min = -0.3

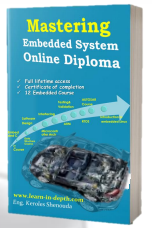
VI_Min = Vss-0.3 V

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

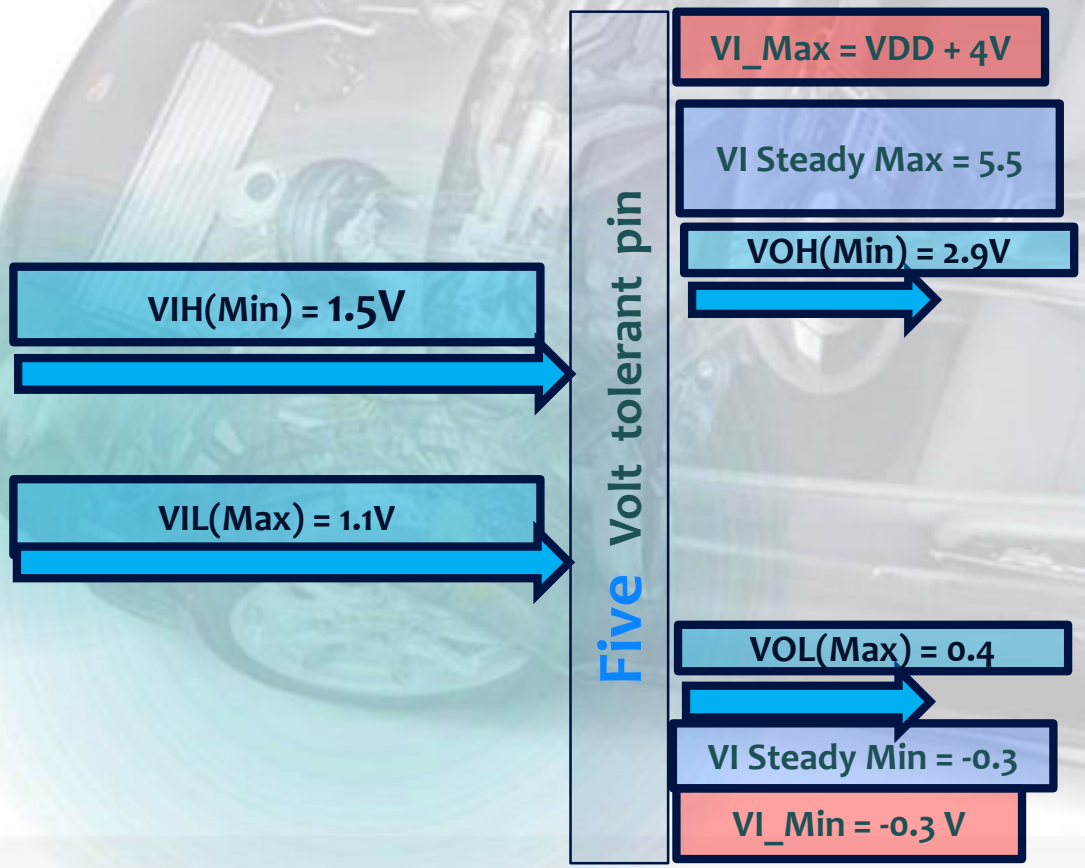
Logic level For Output

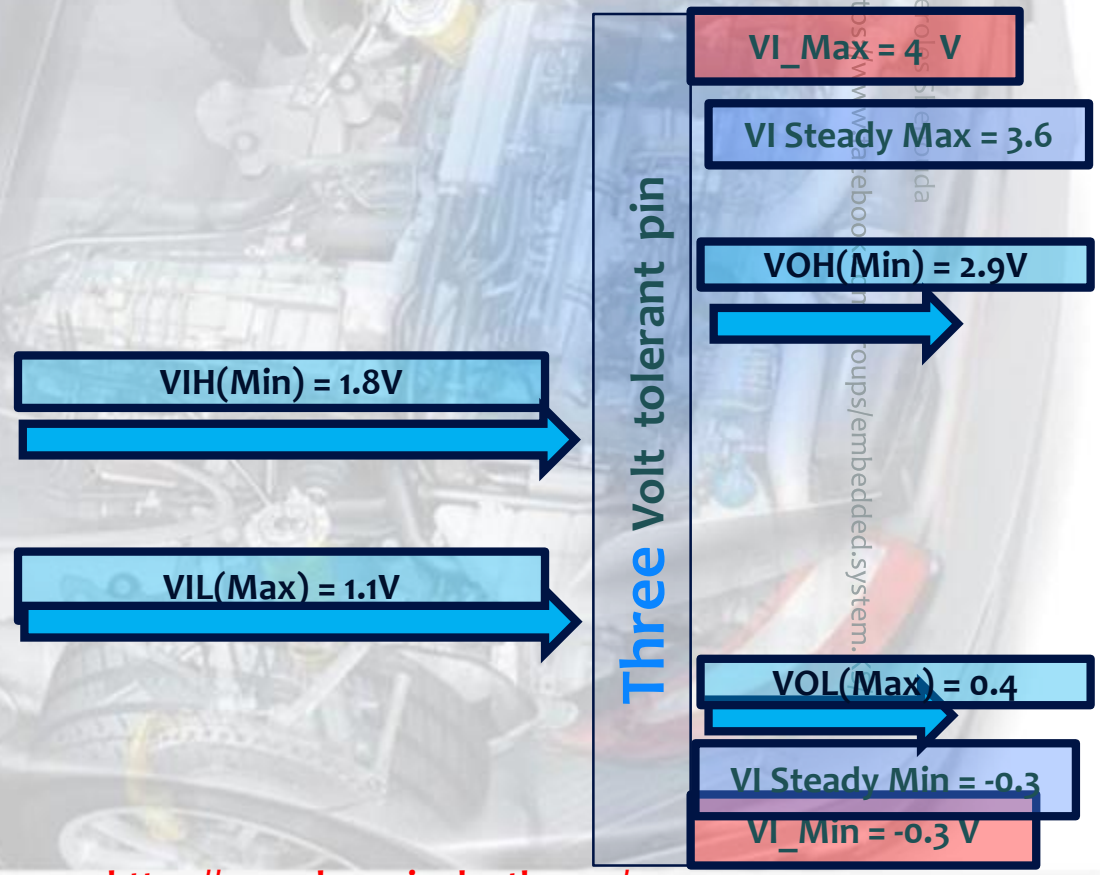
Table 36. Output voltage characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
V_{OL} (1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	CMOS port(2), I_{IO} = +8 mA, 2.7 V < V_{DD} < 3.6 V	-	0.4	
V_{OH} (3)	Output high level voltage for an I/O pin when 8 pins are sourced at same time		V_{DD}–0.4	-	



VDD = 3.3 V

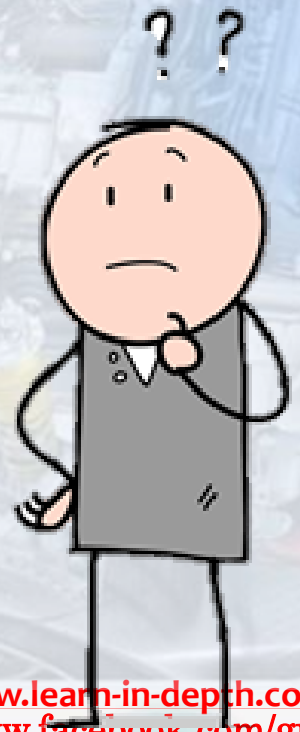




Example 1

- Voltage signal when transmitting Ox32 and Ox3C via UART
- (1 start bit, 1 stop bit, 8 data bits, no parity, baud rate= 9,600)

Think In Depth Draw the frame with Logic level and Time

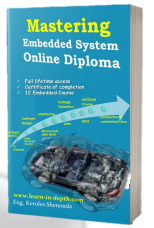


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Example 1

- the binary value of 0x32 is 0b00110010, and the bit **sequence seen on the transmission (TX) line is 01001100.**
- The baud rate is set to 9,600, and thus each bit takes approximately ***0.104ms***.
- When the TX line is idle, the voltage on it is 3V.
- The start bit has OV while the stop bit has 3V

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



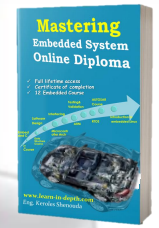
Clock_T = 1/9600 = 0.104 ms

Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



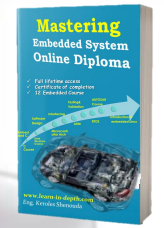
Clock_T = 1/9600 = 0.104 ms

Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

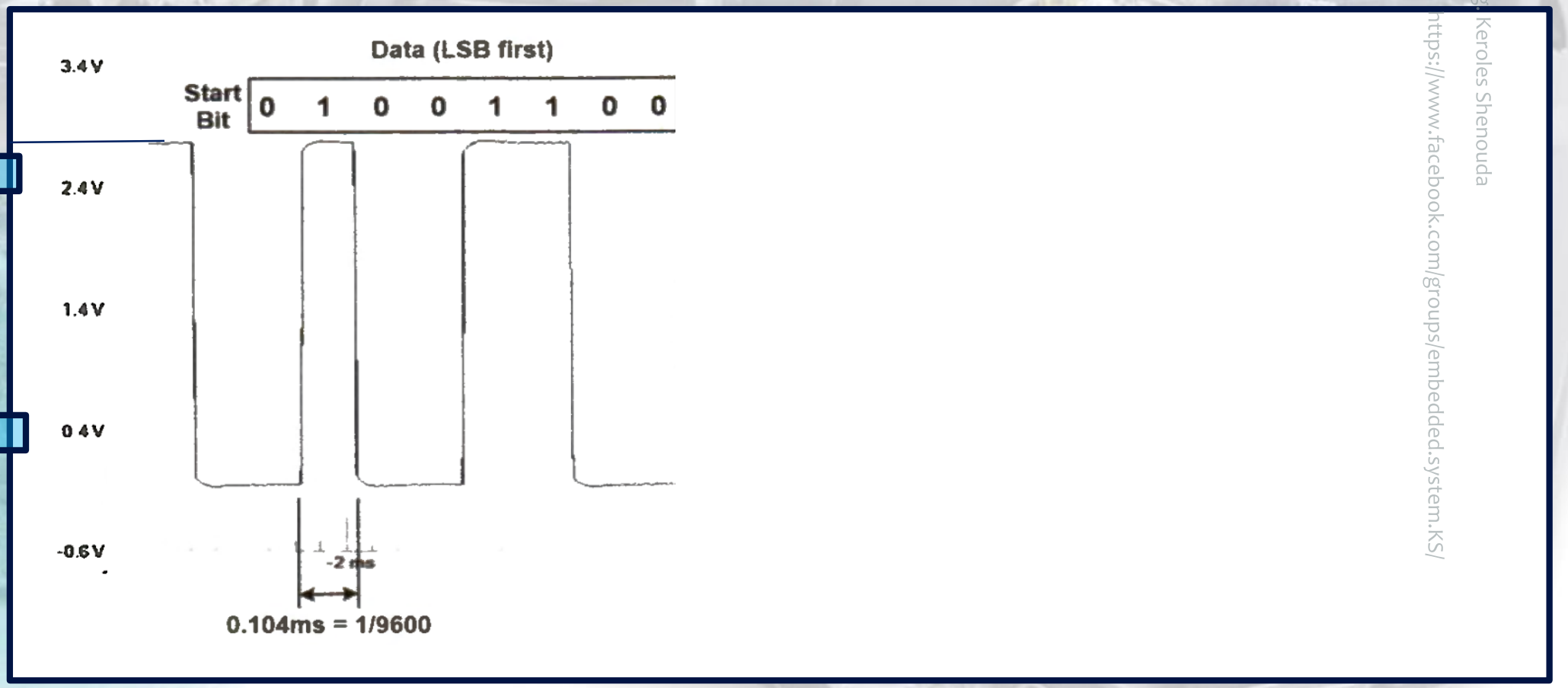


Clock_T = 1/9600 = 0.104 ms

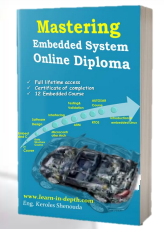
Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

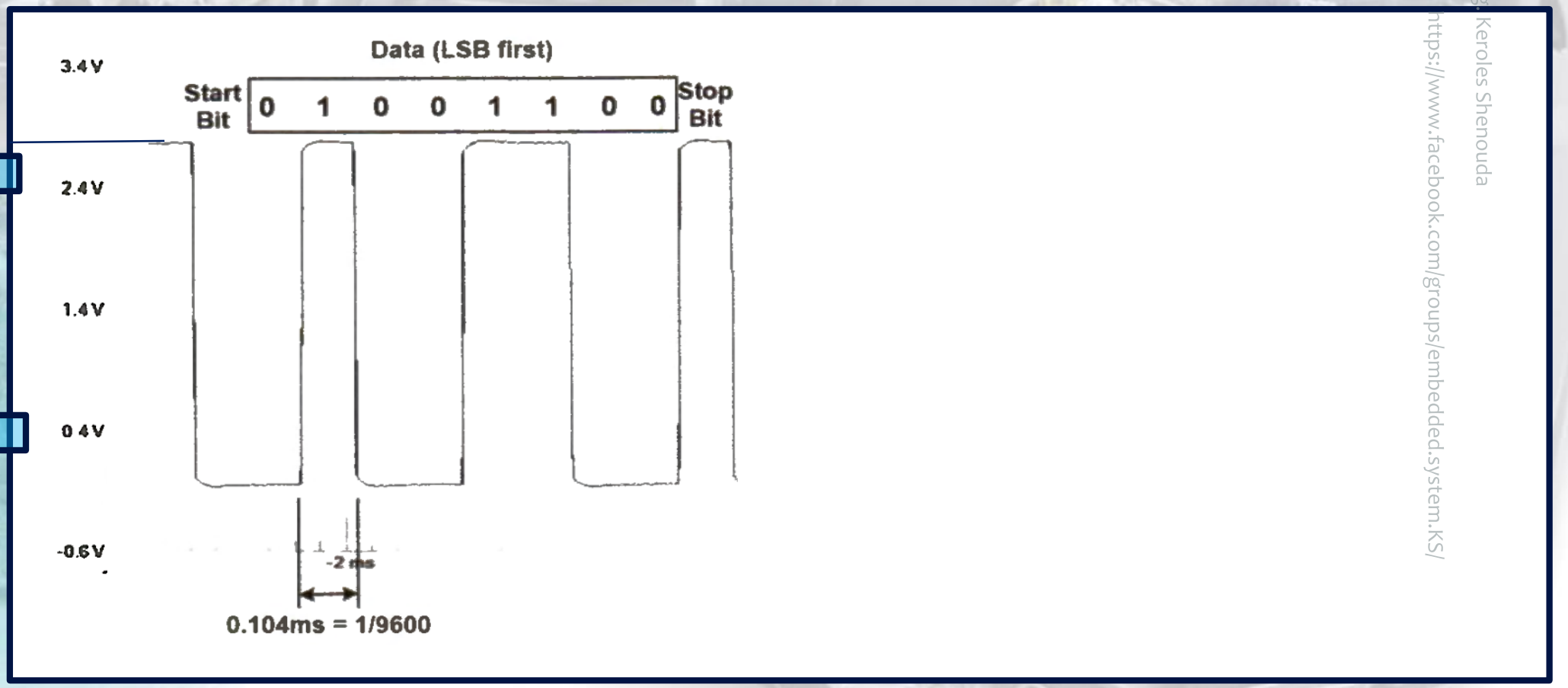


Clock_T = 1/9600 = 0.104 ms

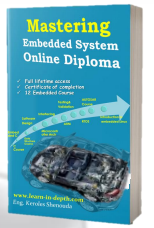
Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

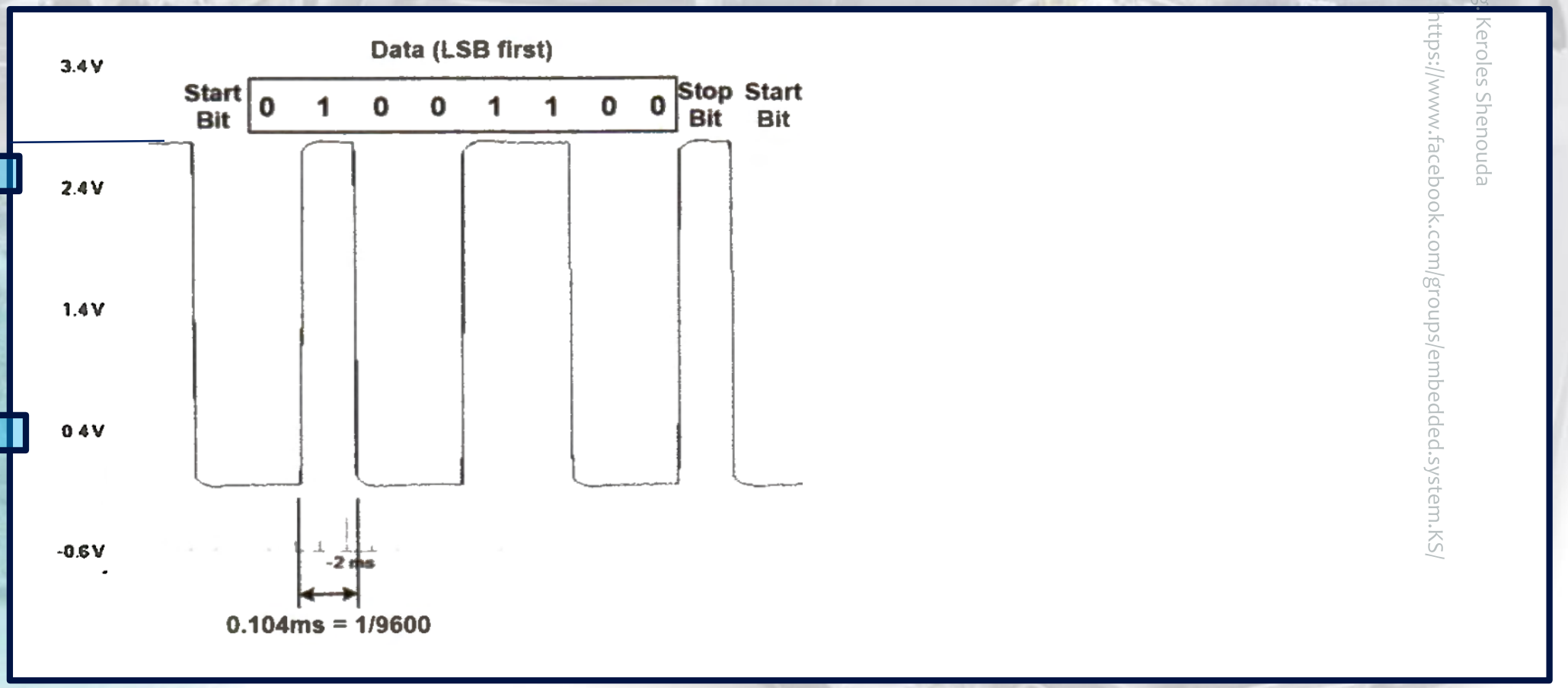


Clock_T = 1/9600 = 0.104 ms

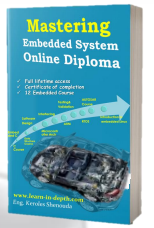
Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

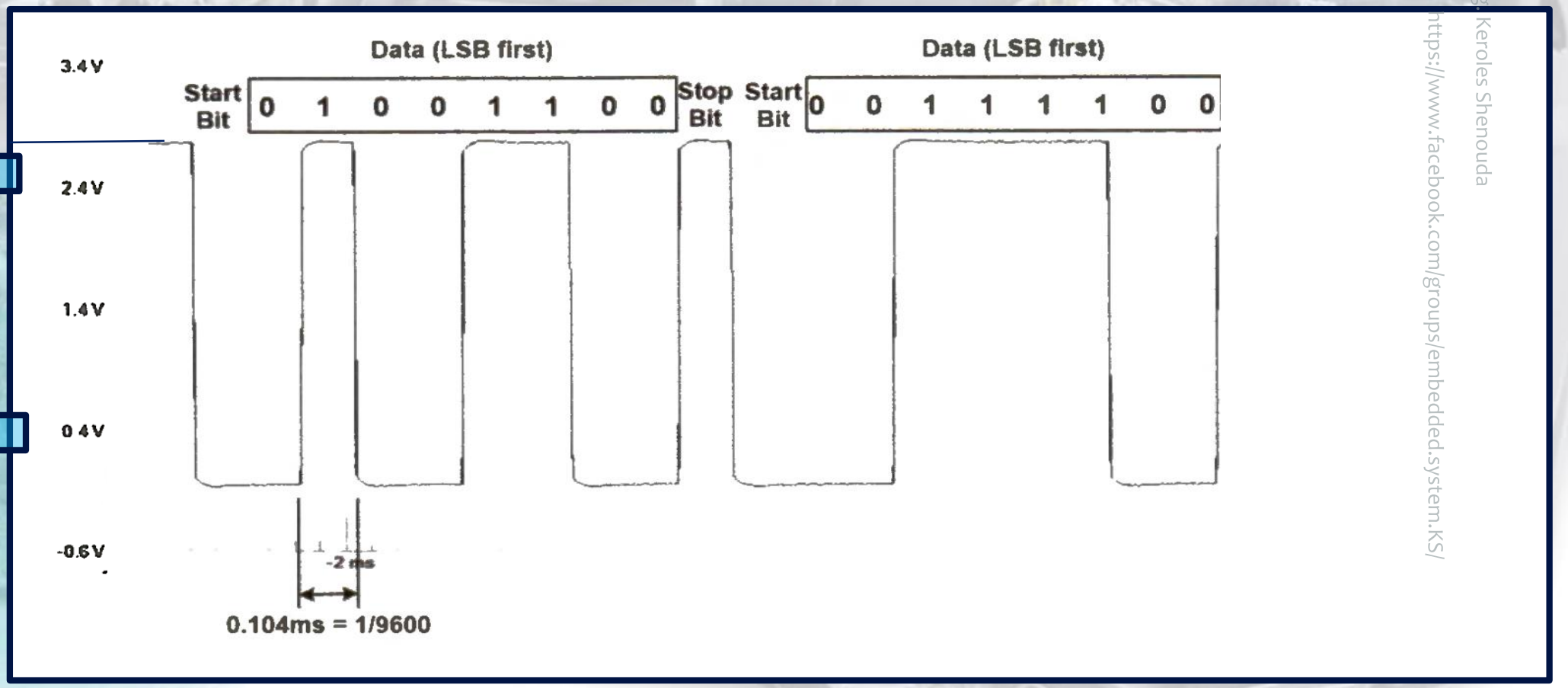


Clock_T = 1/9600 = 0.104 ms

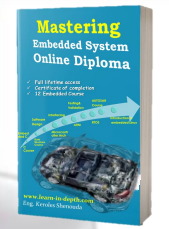
Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

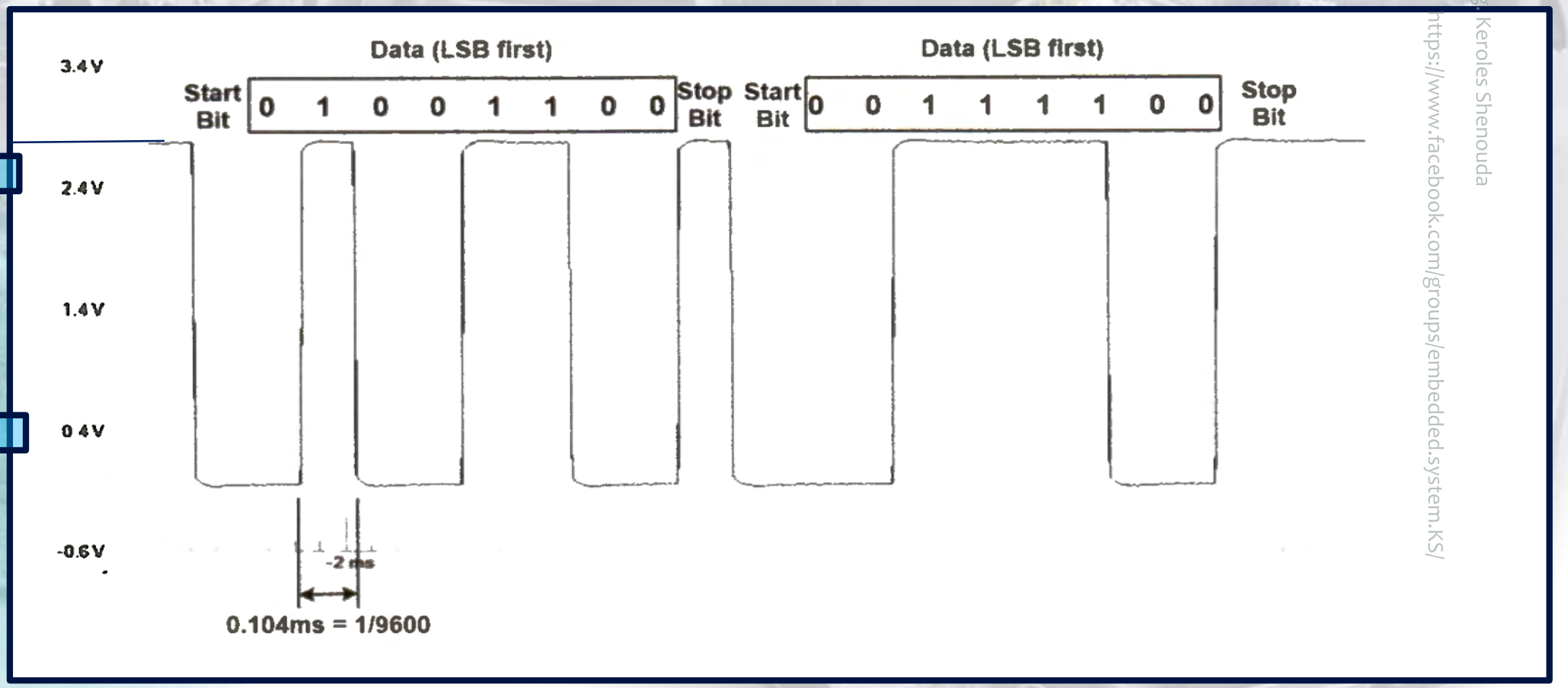


Clock_T = 1/9600 = 0.104 ms

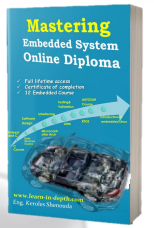
Example 1

0x32 == 0 0 1 1 0 0 1 0

0x3C == 0 0 1 1 1 1 0 0



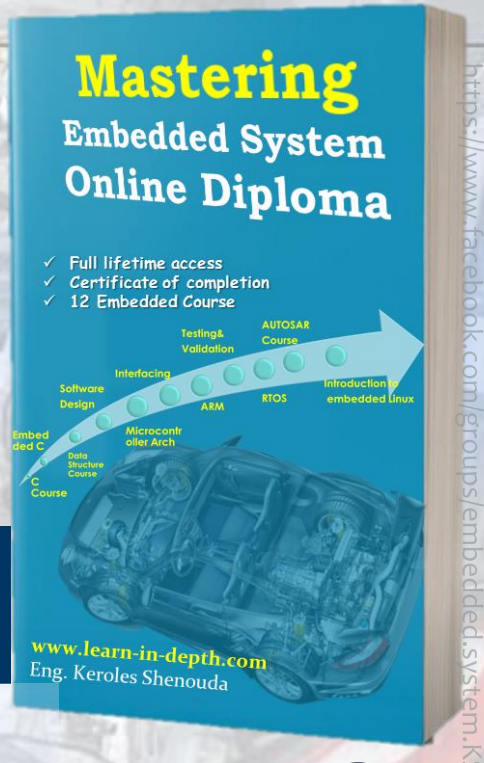
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

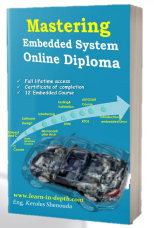
#Be_professional_in_embedded_system

NRZ (non-return-to-zero)



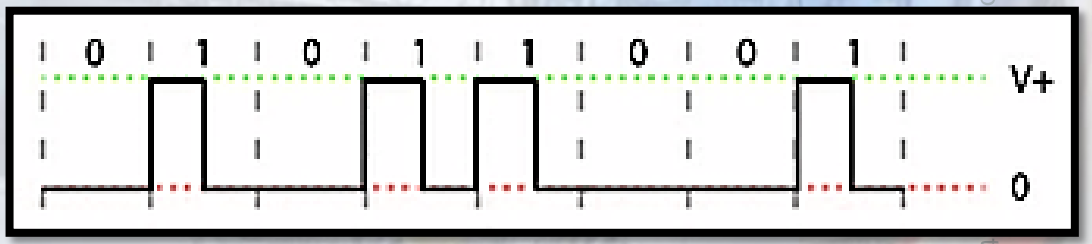
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



RZ (return-to-zero)

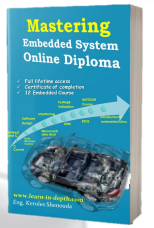
- RZ (return-to-zero) refers to a form of digital data transmission in which the binary low and high states, represented by numerals 0 and 1, are transmitted by voltage pulses having certain characteristics. The signal state is determined by the voltage during the first half of each data binary digit . The signal returns to a resting state (called zero) during the second half of each bit.



Examples are:

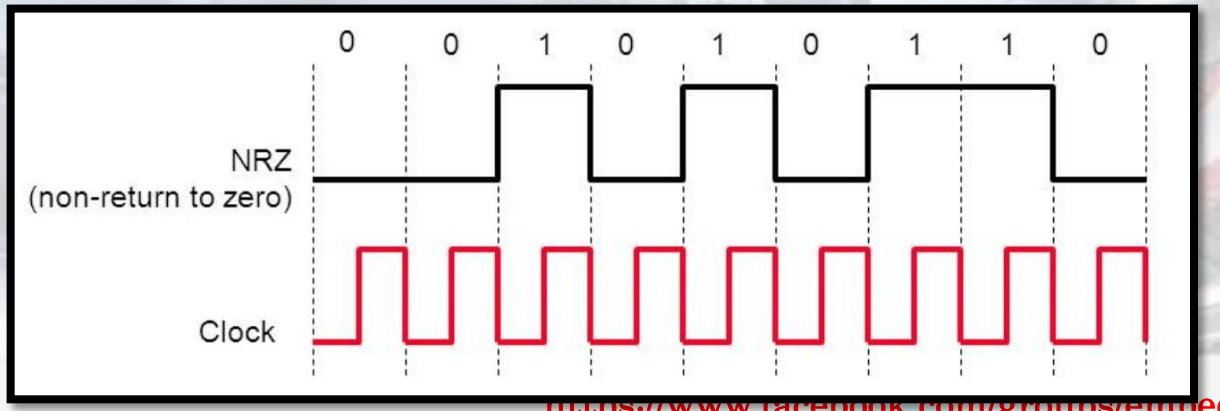
Logic 0 = 0 volts for 1 bit

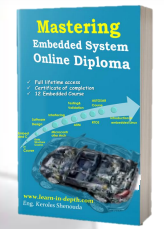
Logic 1 = +5 volts for 1/2 bit, then 0 volts for 1/2 bit



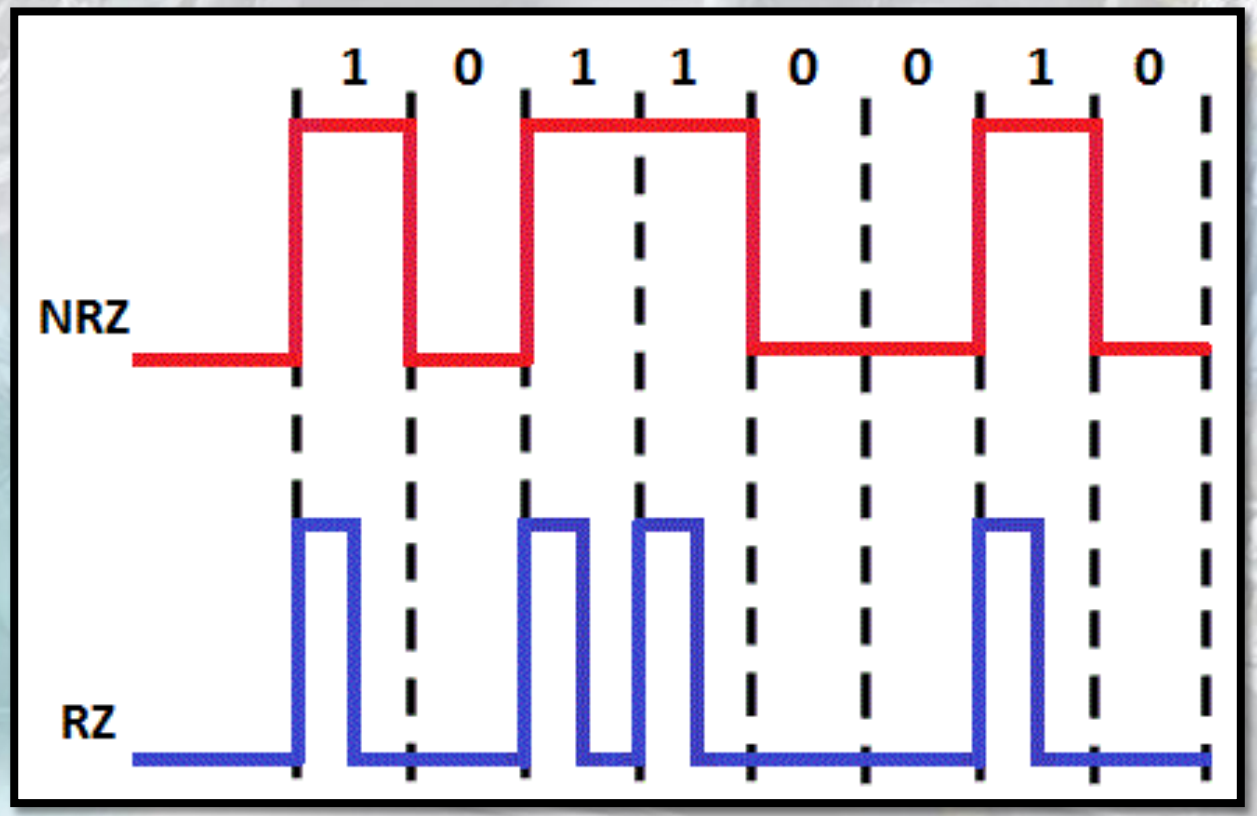
NRZ (non-return-to-zero)

- RZ (non-return-to-zero) refers to a form of digital data transmission in which the binary low and high states, represented by numerals 0 and 1, are transmitted by specific and constant DC (direct-current) voltage s.
- In positive-logic NRZ, the low state is represented by the more negative or less positive voltage , and the high state is represented by the less negative or more positive voltage. Examples are:
- Logic 0 = +0.5 volts
- Logic 1 = +5.0 volts

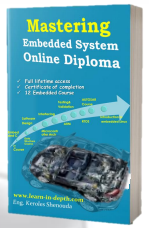




NRZ Vs RZ



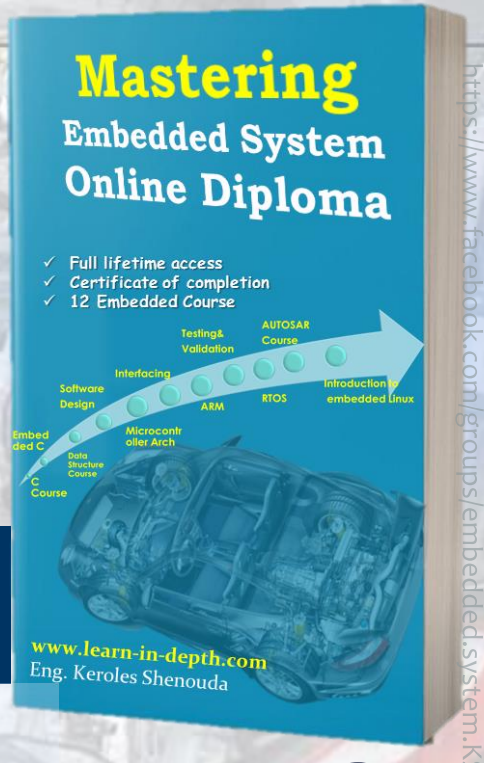
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

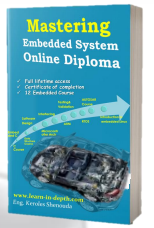
#Be_professional_in_embedded_system

UART Over Sampling Mechanism



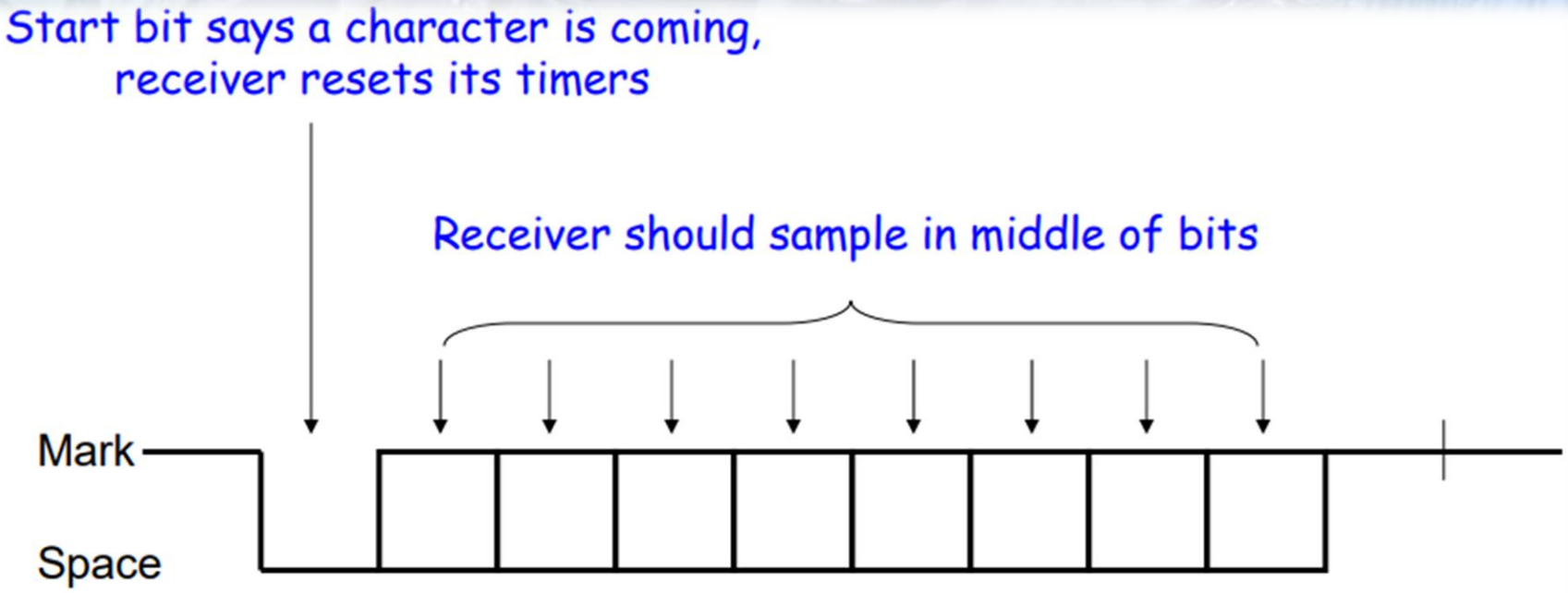
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

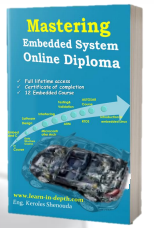


UART Character Reception

- Receiver uses a timer (counter) to time when it samples.
- **Transmission rate (i.e., bit width) must be known!**

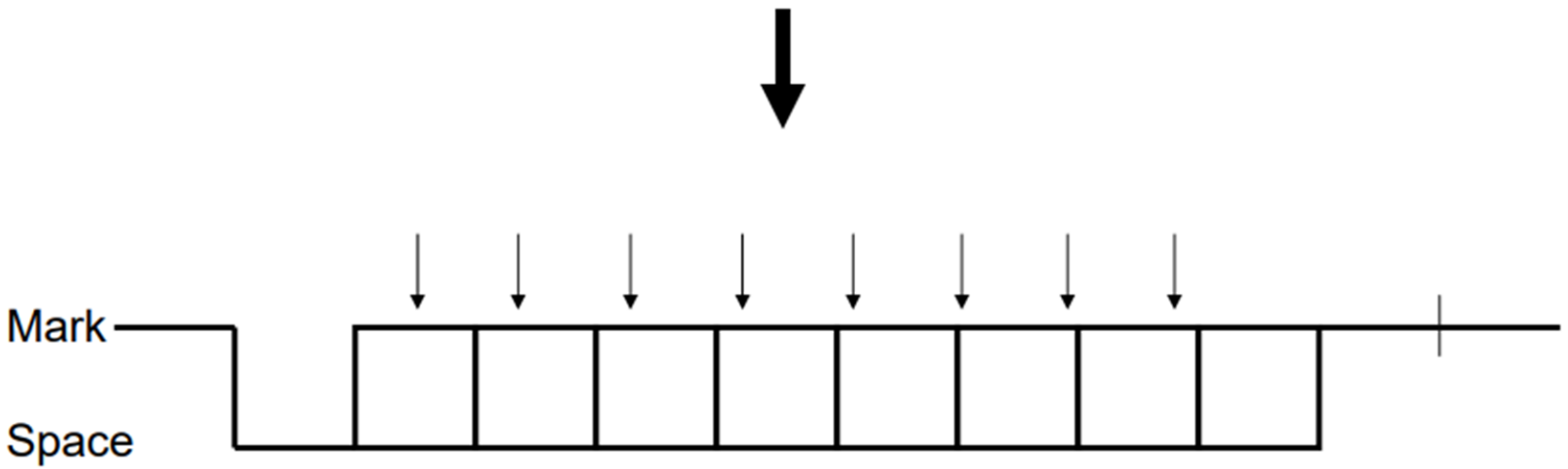


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

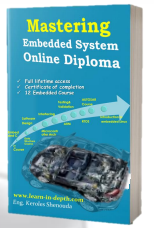


UART Character Reception

If receiver samples too quickly, see what happens…

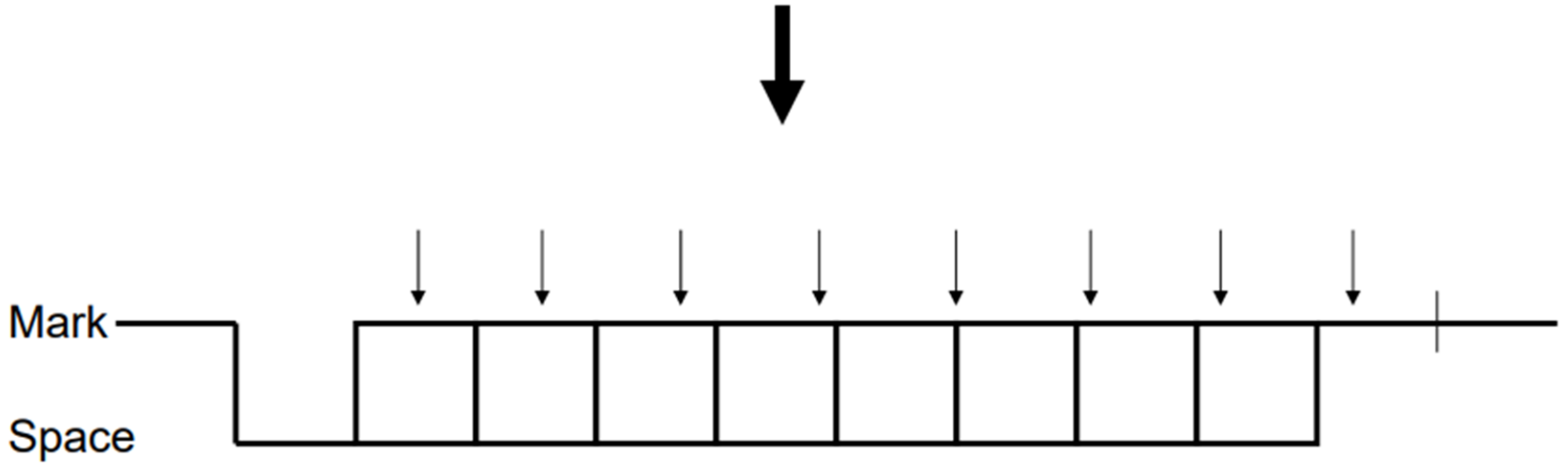


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



UART Character Reception

If receiver samples too slowly, see what happens…

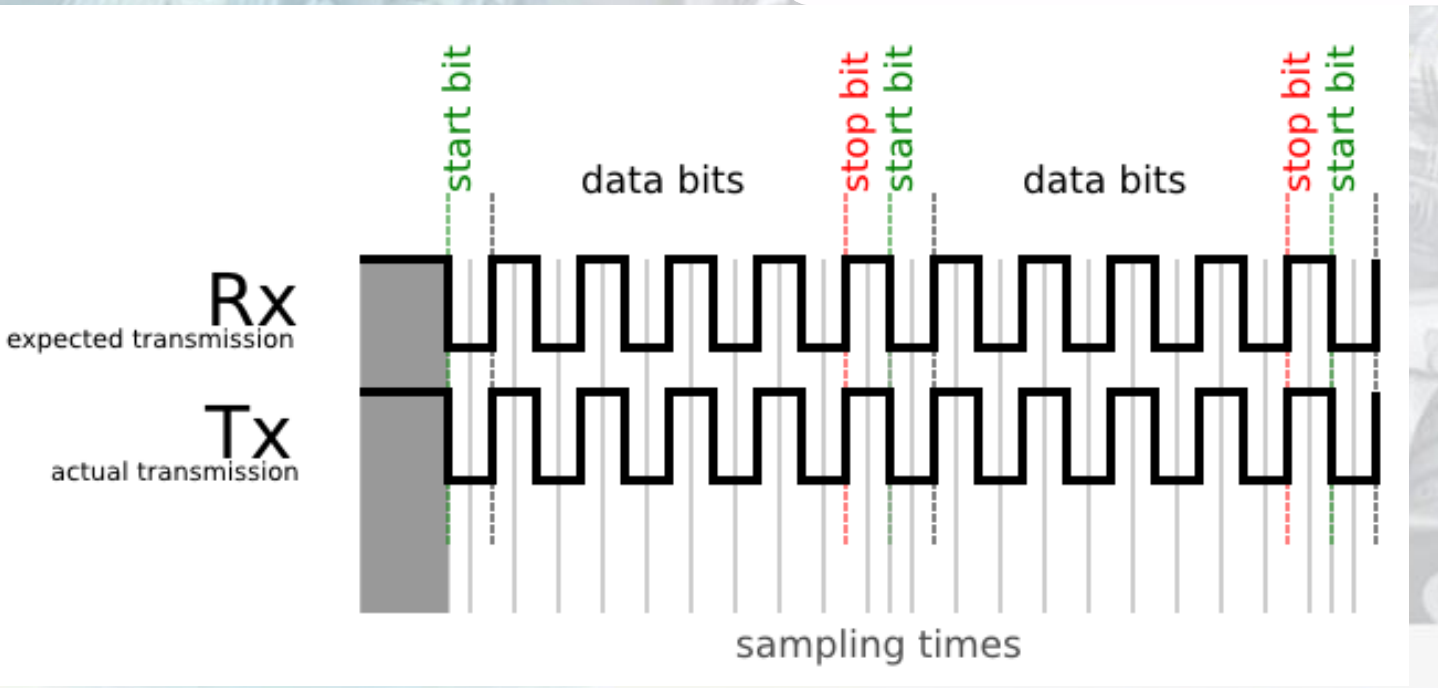


Receiver resynchronizes on every start bit.
Only has to be accurate enough to read 9 bits.

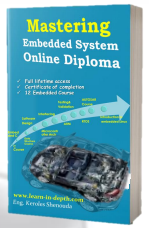
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

So to keep the receiver Sampling in middle of bits

Receiver and Transmitter Need to be on the Same Baud rate / Bitrate

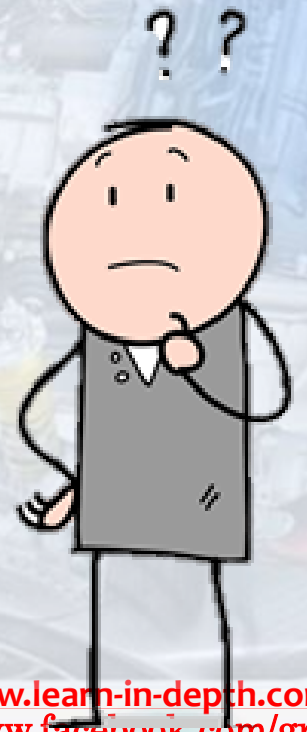


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

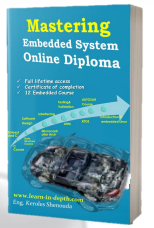


the Normal Sampling Example
In case BaudRate = 9600 bps

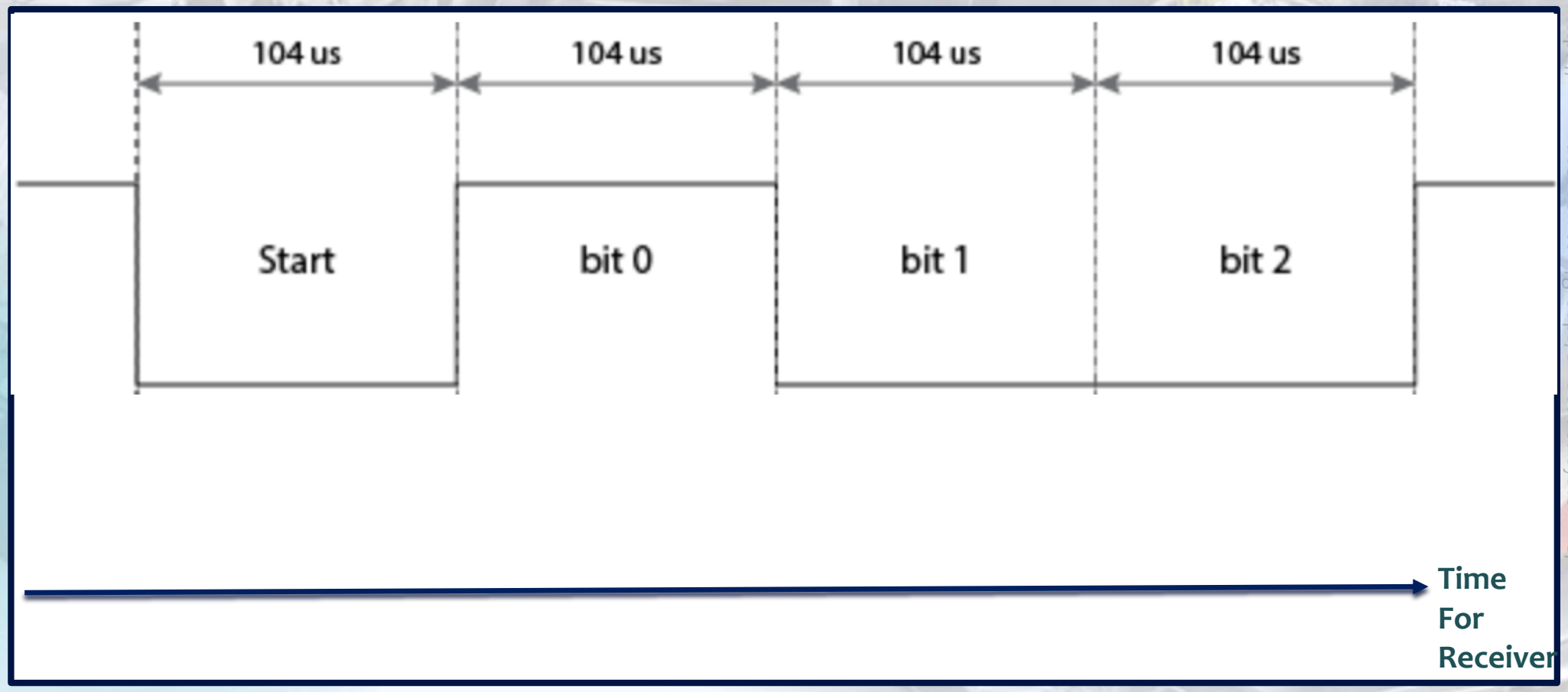
Think In Depth Draw the frame with Logic level and Time



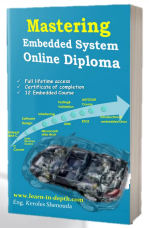
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



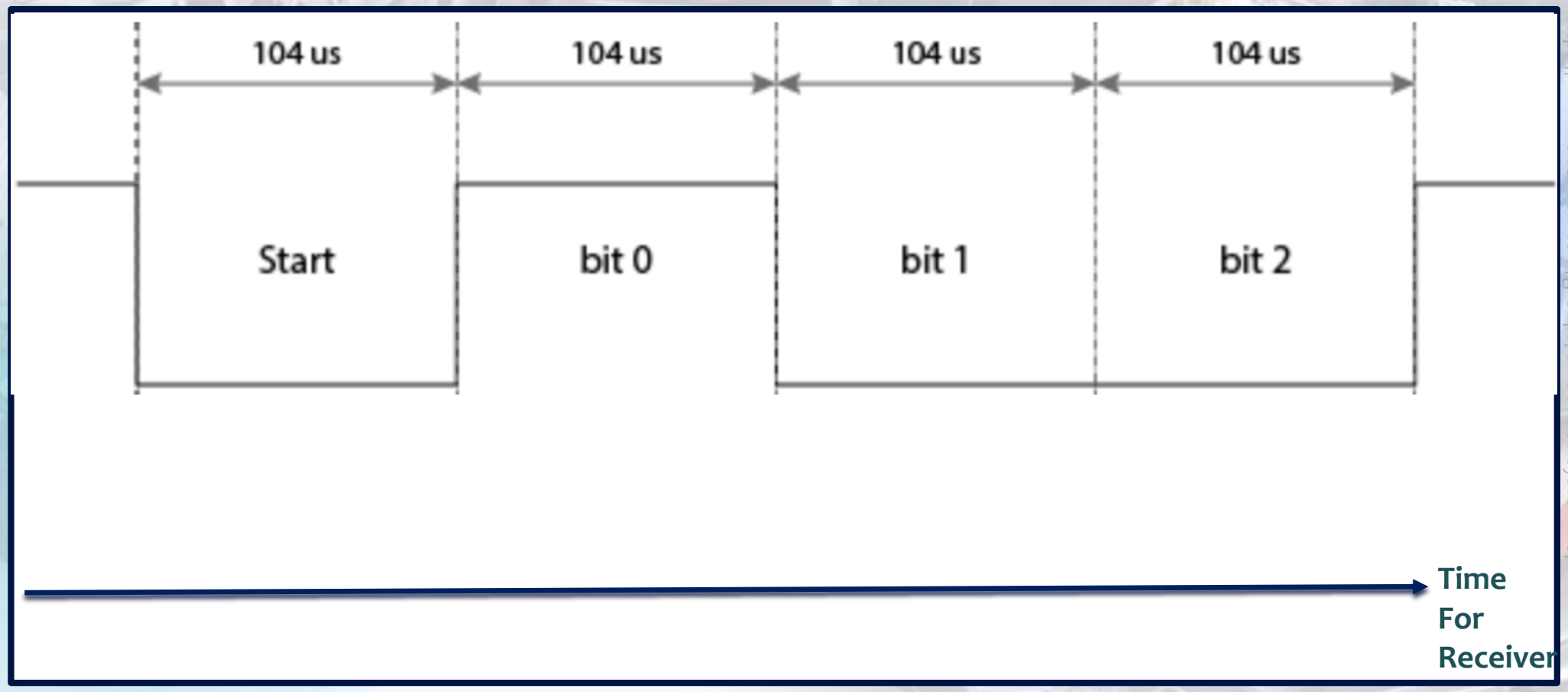
the Normal Sampling Example
In case BaudRate = 9600 bps



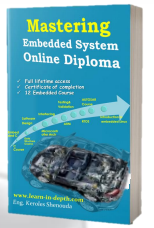
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



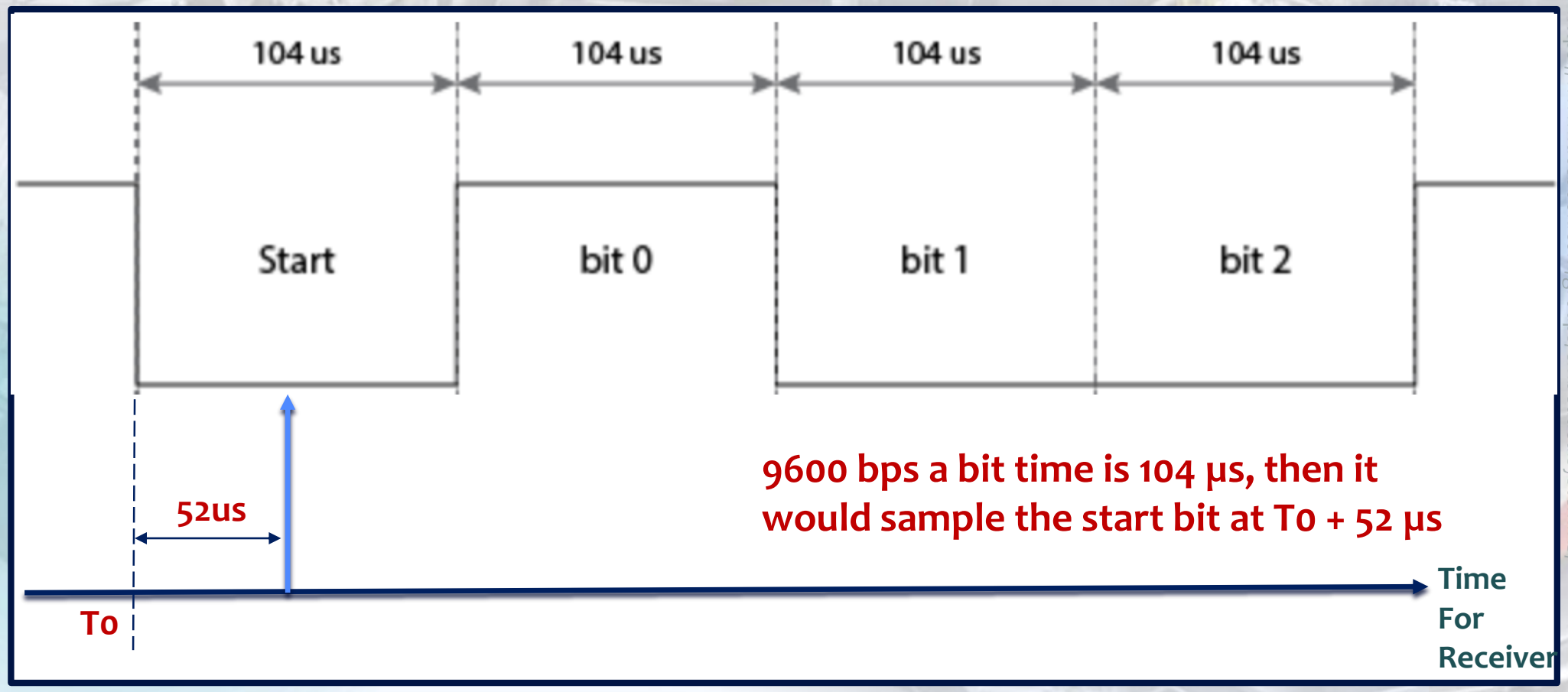
the Normal Sampling Example
In case BaudRate = 9600 bps



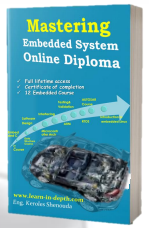
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



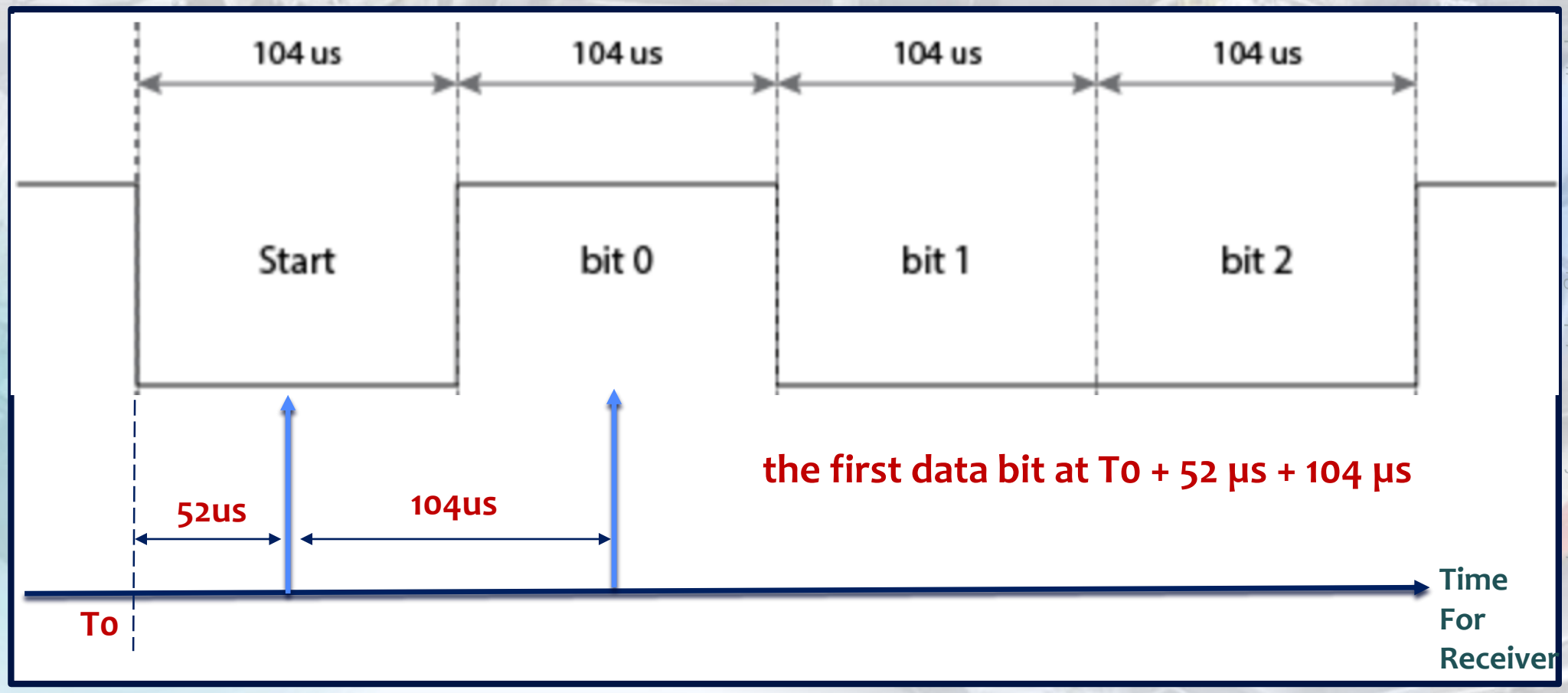
the Normal Sampling Example
In case BaudRate = 9600 bps



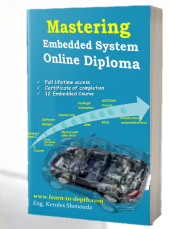
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



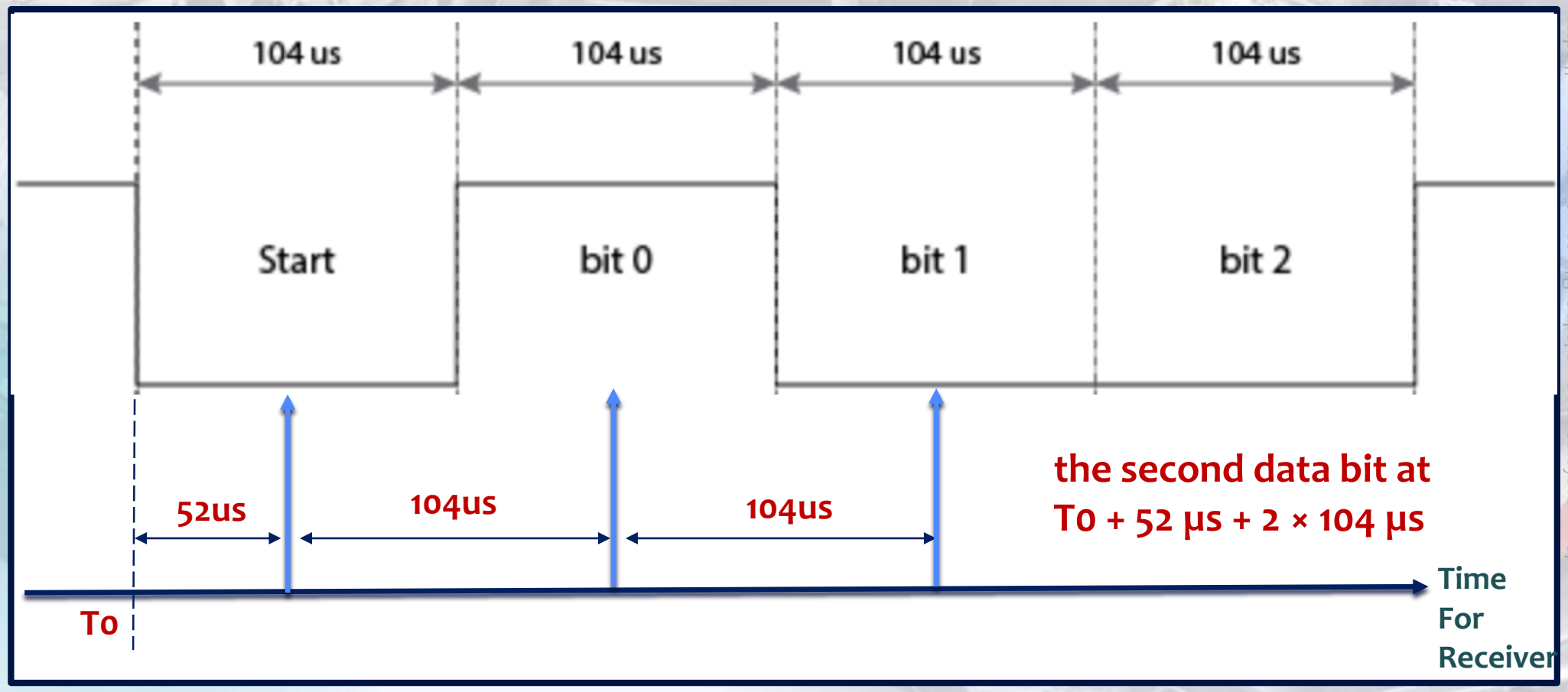
the Normal Sampling Example
In case BaudRate = 9600 bps



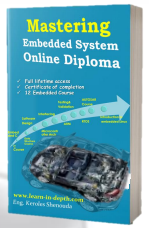
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



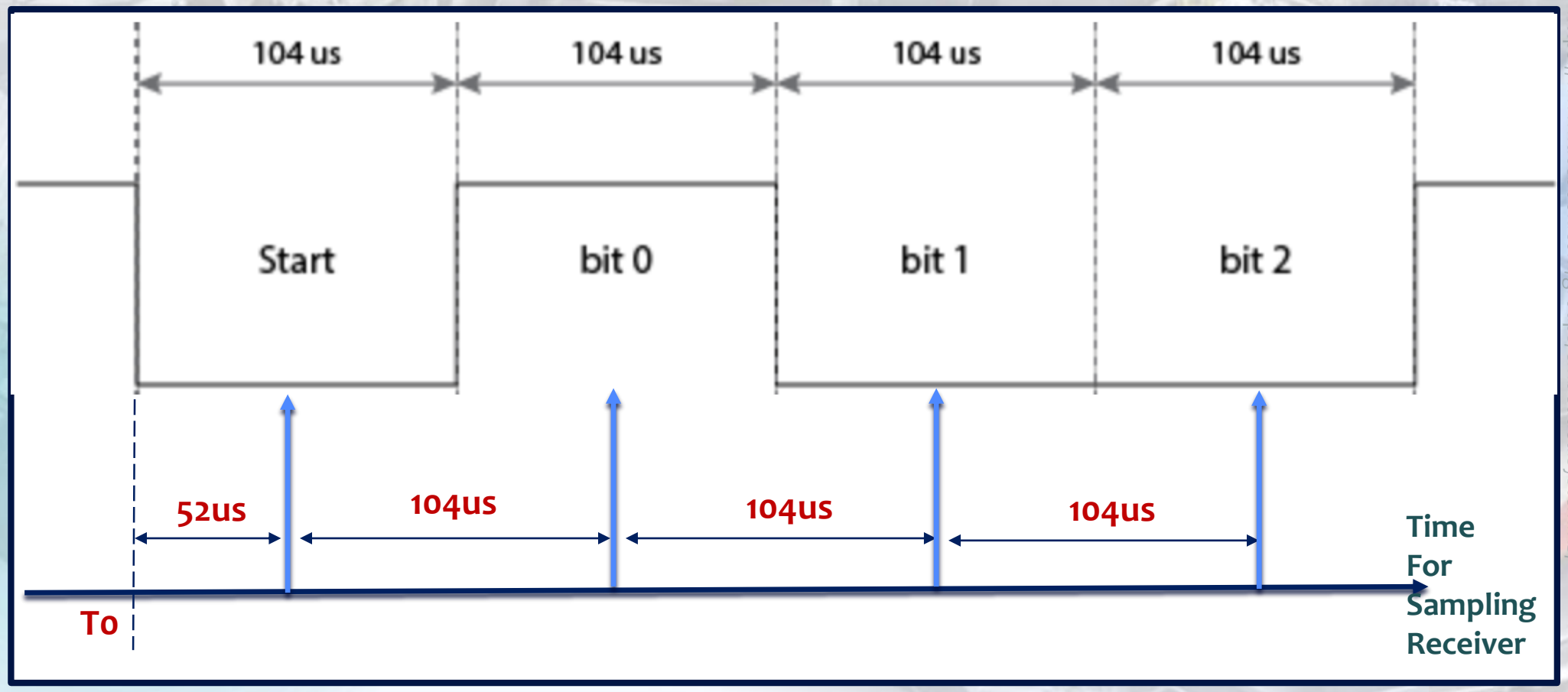
the Normal Sampling Example
In case BaudRate = 9600 bps



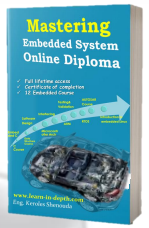
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



the Normal Sampling Example
In case BaudRate = 9600 bps



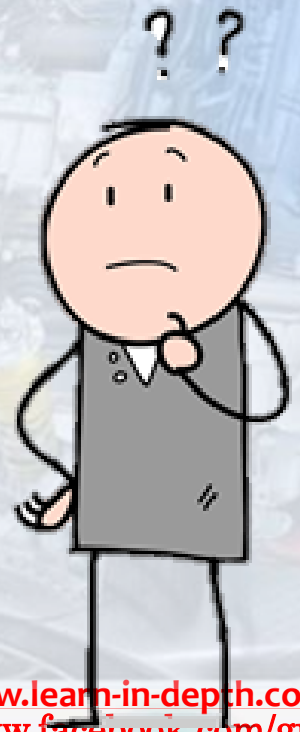
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



the Normal Sampling Example In case BaudRate = 9600 bps

- For a 52 µs timing you need twice the 9600 bps clock frequency, or 19200 Hz. But this is only a basic detecting method

What About More advanced (more accurate) methods



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

the Normal Sampling Example
In case BaudRate = 9600 bps

- More advanced (more accurate) methods will take several samples in a row, **to avoid hitting just that one spike.**
- Then you may indeed need a 16 × 9600 Hz clock to get 16 ticks per bit
- Or need a 8 × 9600 Hz clock to get 8 ticks per bit
- of which you may use, say, 5 or so in what should be the middle of a bit.
- And the use a voting system to see whether it should be read as high or low.

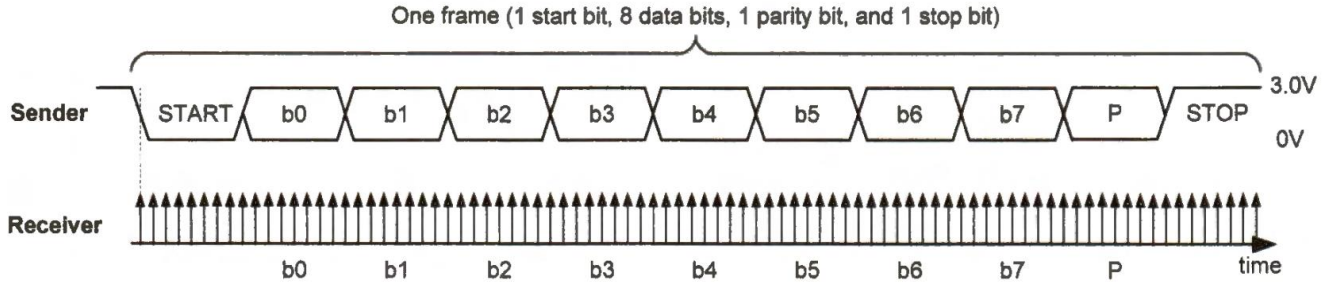
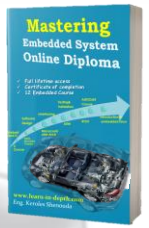


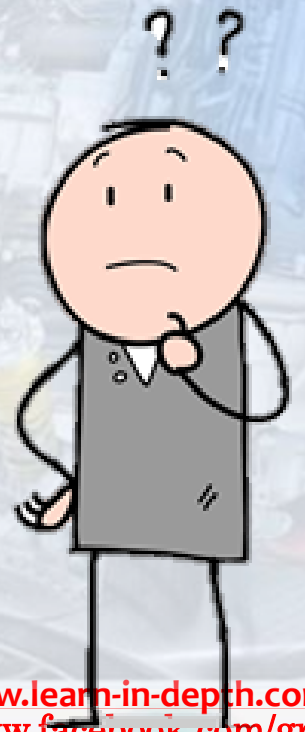
Figure 22-3. Receiver oversamples each bit 8 times

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

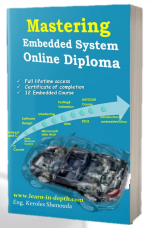


the Normal Sampling Example
In case BaudRate = 9600 bps

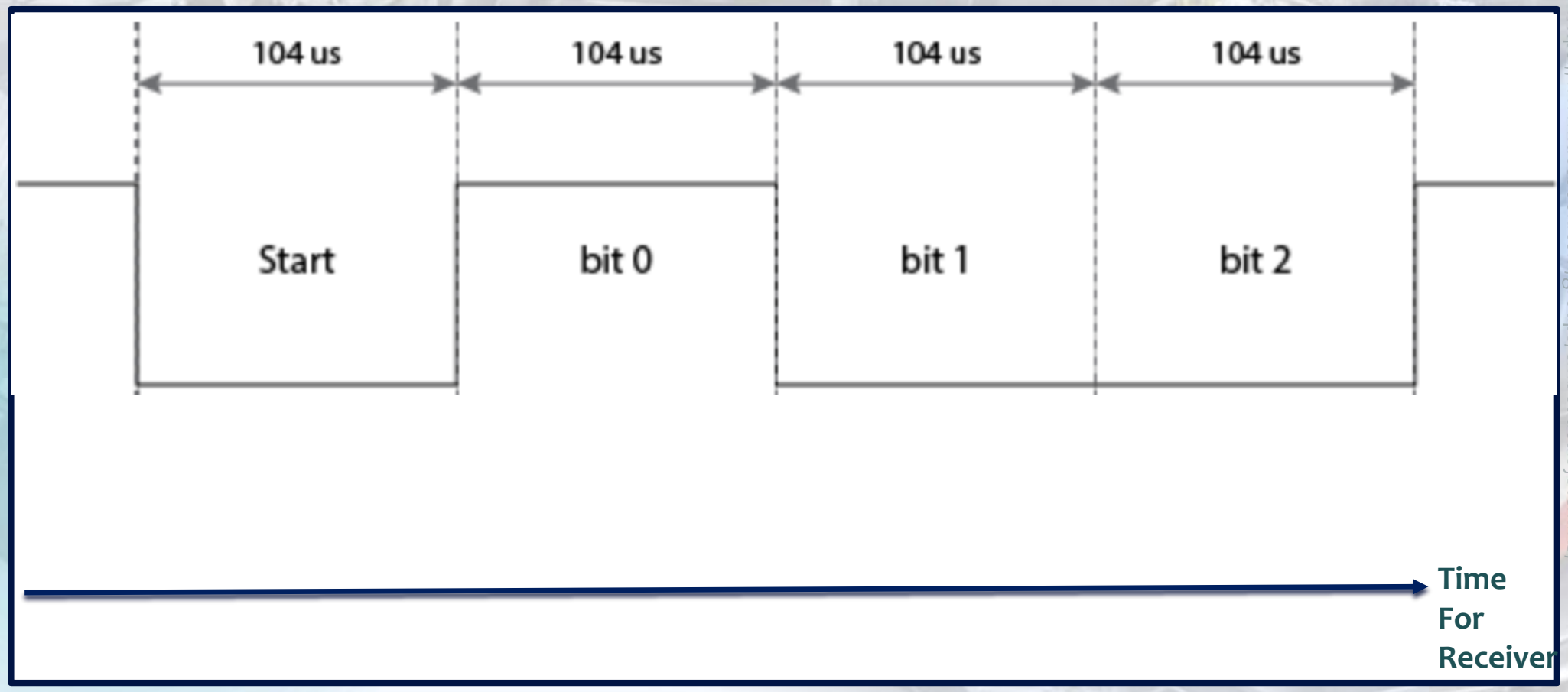
What That Means in your Example
Apply oversamples each bit 8 times



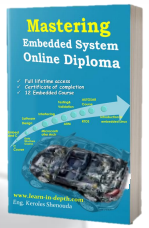
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



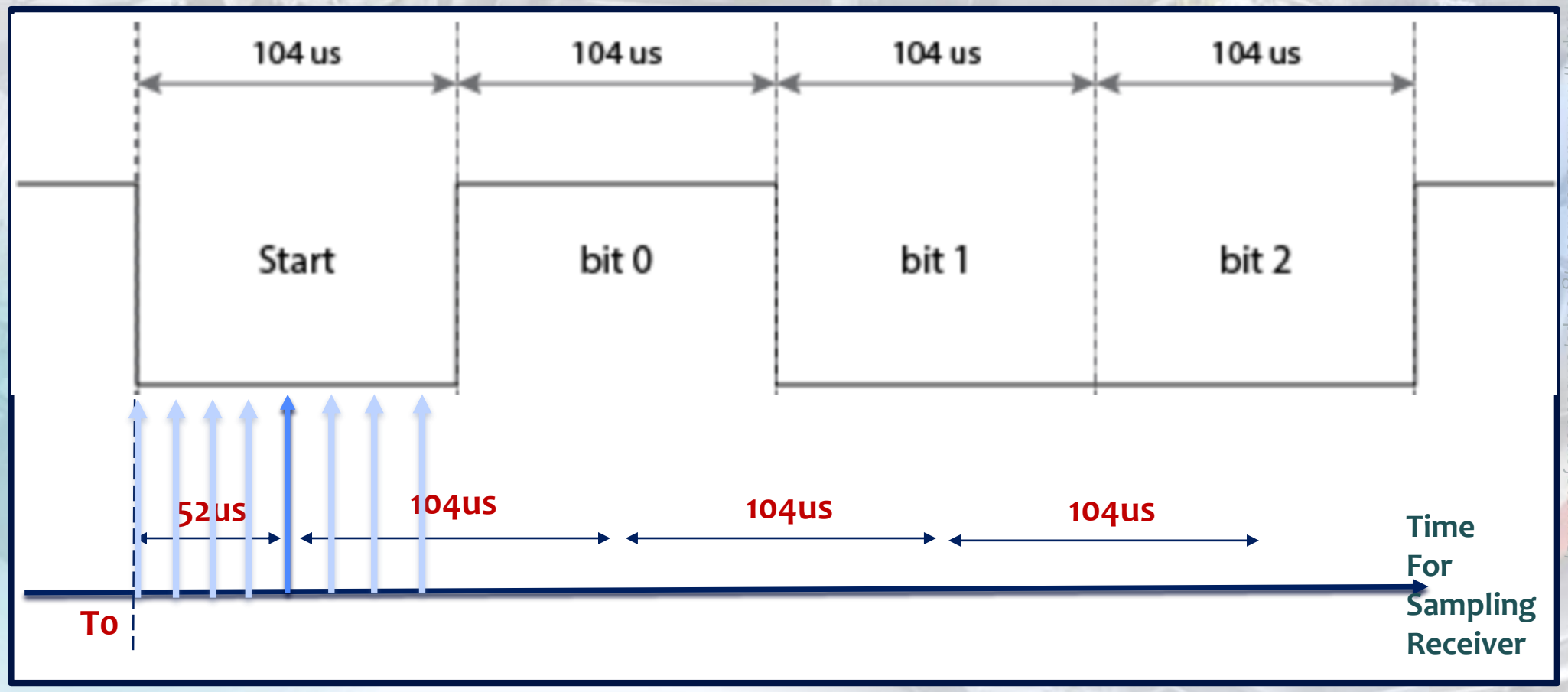
the Normal Sampling Example
In case BaudRate = 9600 bps



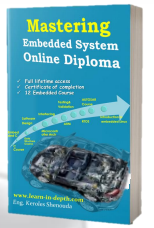
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



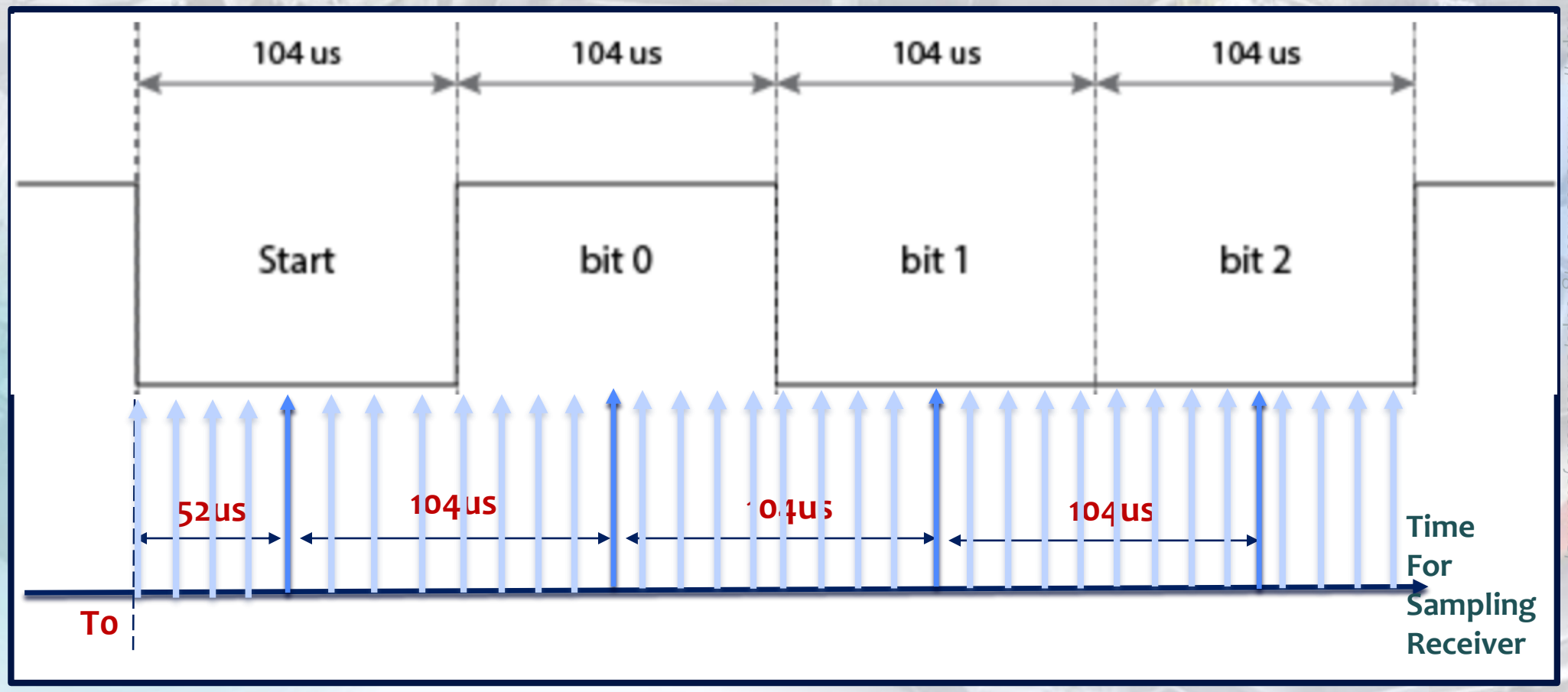
the Normal Sampling Example In case BaudRate = 9600 bps



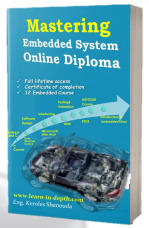
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



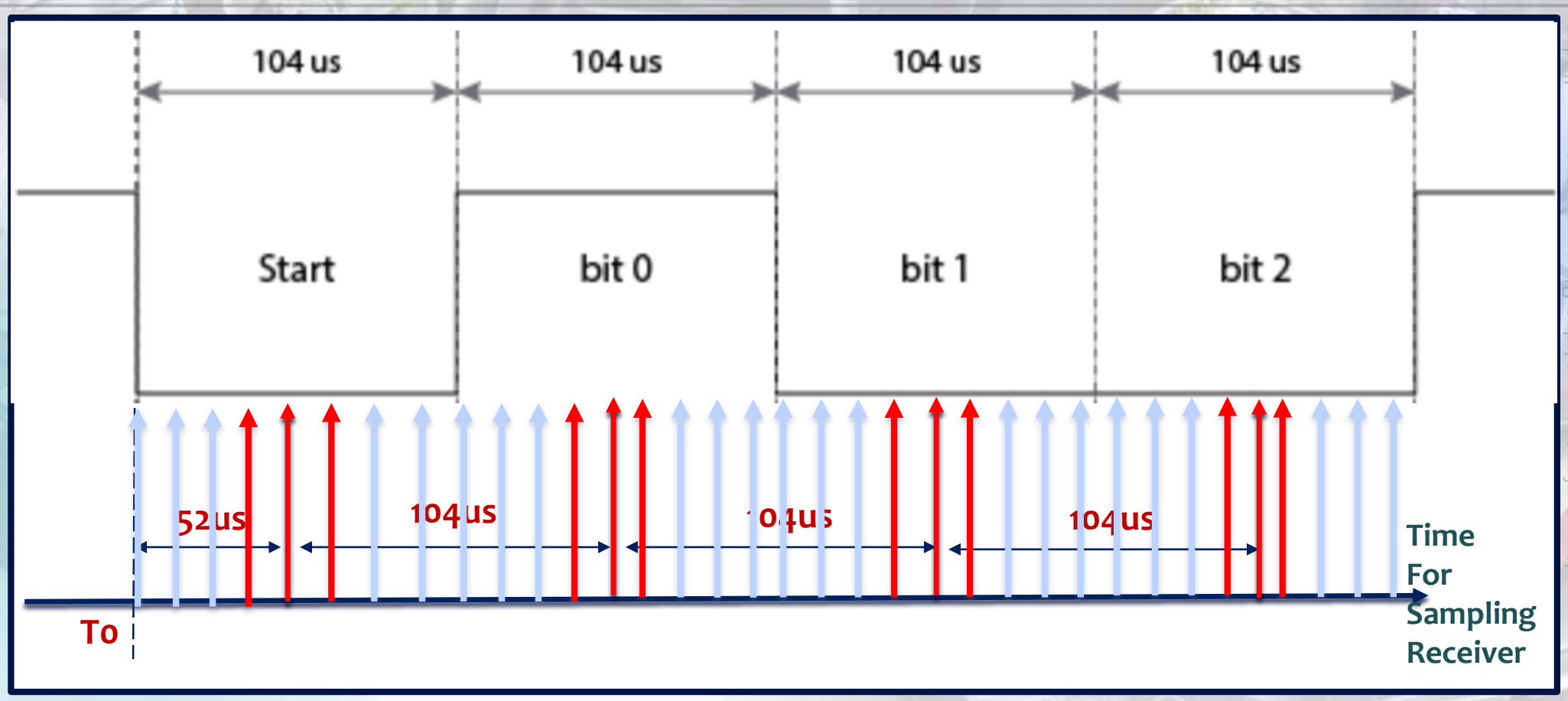
sampling clock = 8 * 9600 Hz = 76800 Hz clock to get 8 ticks per



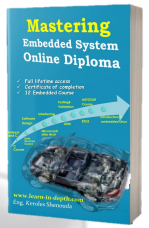
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



3 or so in what should be the middle of a bit And the use a voting system to see whether it should be read as high or low.

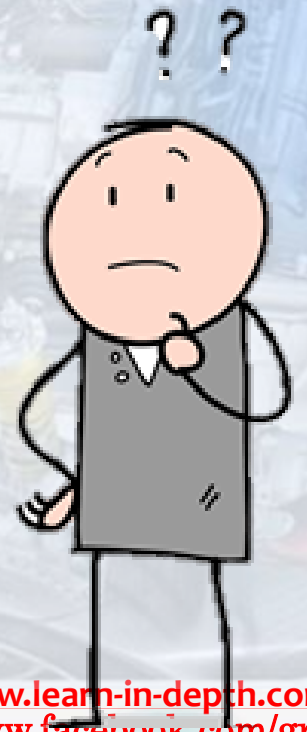


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

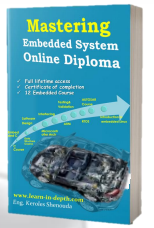


the Normal Sampling Example
In case BaudRate = 9600 bps

What That Means in your Example
Apply oversamples each bit 16 times

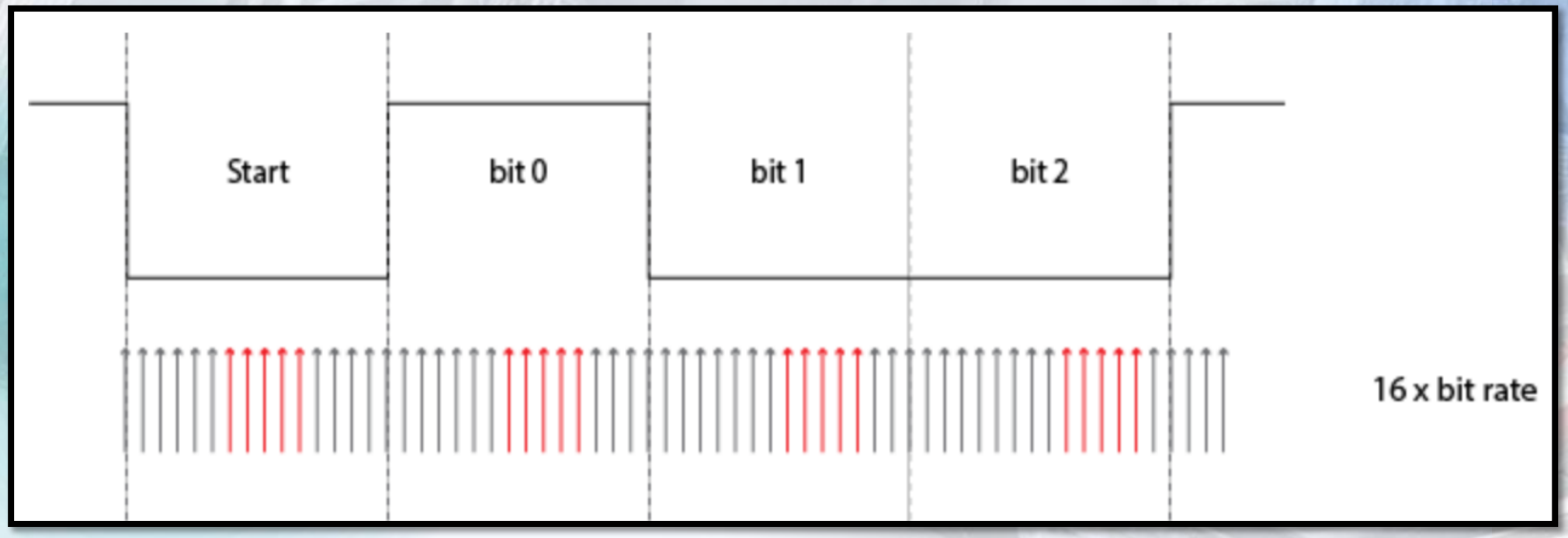


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

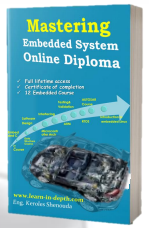


sampling clock to 153 600 Hz
= 16 × 9600 Hz clock to get 16 ticks per bit

- 5 or so in what should be the middle of a bit. And the use a voting system to see whether it should be read as high or low.



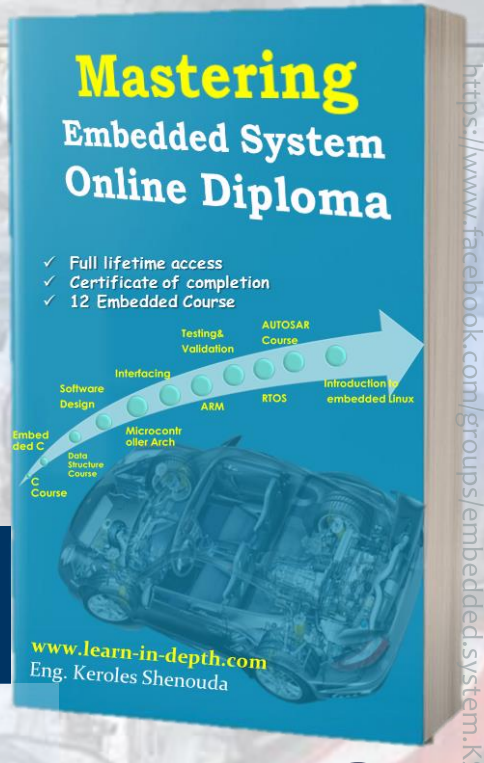
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

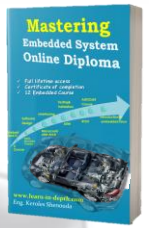
#Be_professional_in_embedded_system

send or receive data via UART ports by using three different methods



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



send or receive data via UART ports by using three different methods

- Polling
  - Polling is the simplest but most inefficient method
- Interrupt
  - is more efficient but not suitable for high data transfer rates.
- DMA
  - is complex but the most effective.

Polling Read

27.6.1 Status register (USART_SR)

Address offset: 0x00
Reset value: 0x00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

- When UART receives a byte, hardware sets the receive register not empty flag (RXNE) in the status register (ISR).
- In the polling approach, software constantly checks the RXNE flag and reads the receive data register (RDR) once it is set.
- Reading register DR clears the RXNE flag automatically.

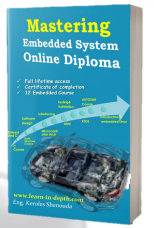
Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_DR register. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. This clearing sequence is recommended only for multibuffer communication.

0: Data is not received

1: Received data is ready to be read.

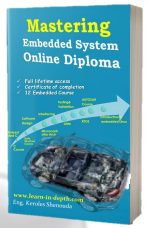
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Polling write

- When UART sends a byte, software must wait until the TxE (transmission data register empty) flag is set in the status register
- After exiting the *for* loop, software must wait for the transmission complete (TC) flag to ensure the last byte has been sent out.

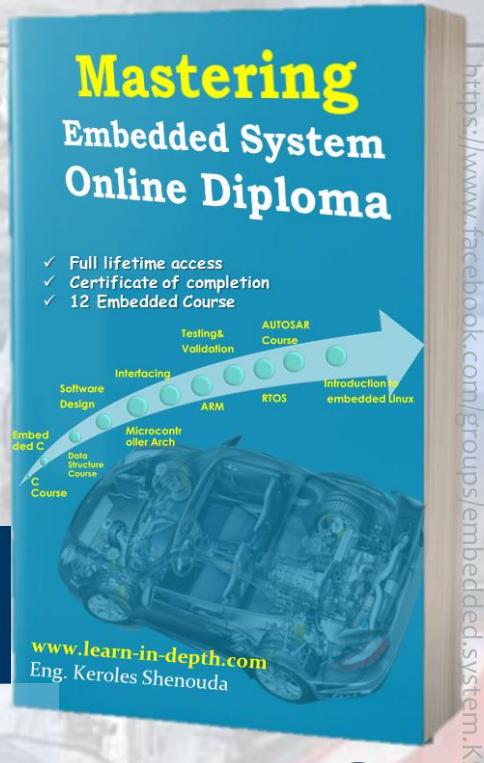
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

UART Communication via Interrupt

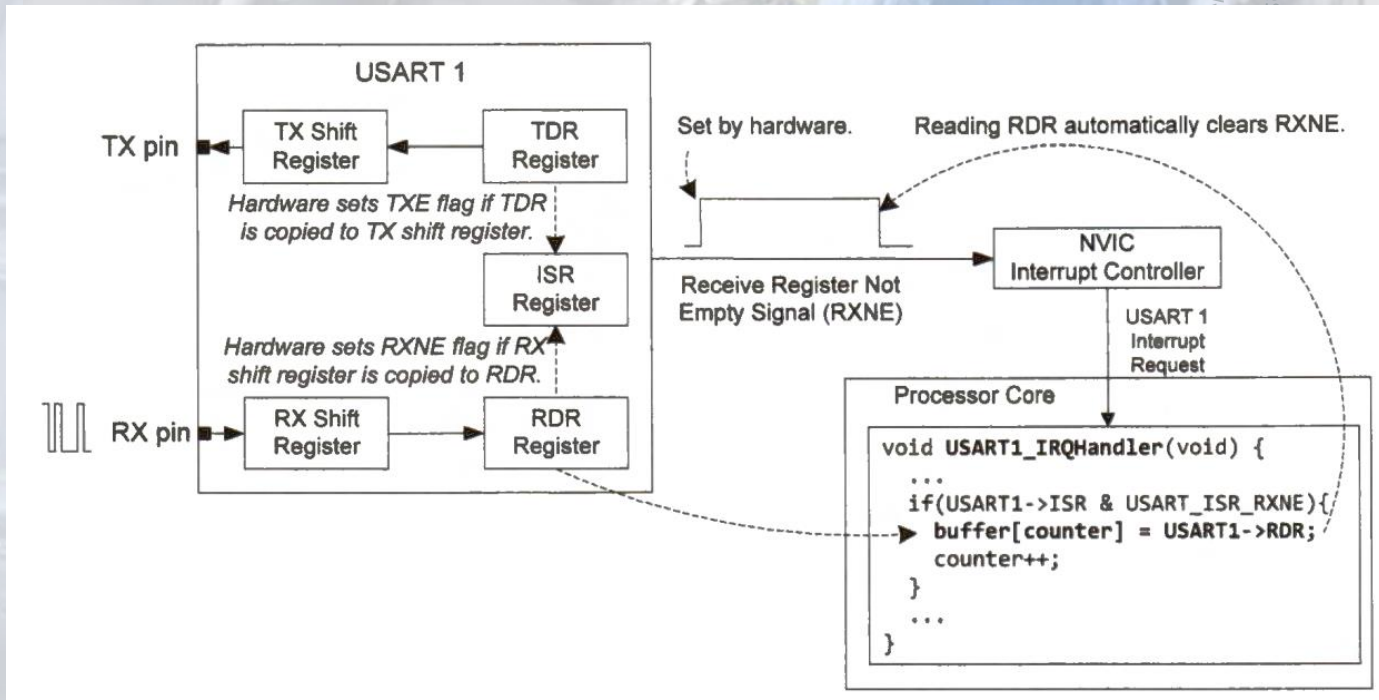


LEARN-IN-DEPTH
Be professional in embedded system

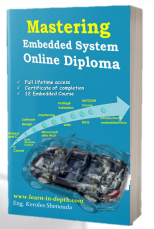
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

UART Communication via Interrupt

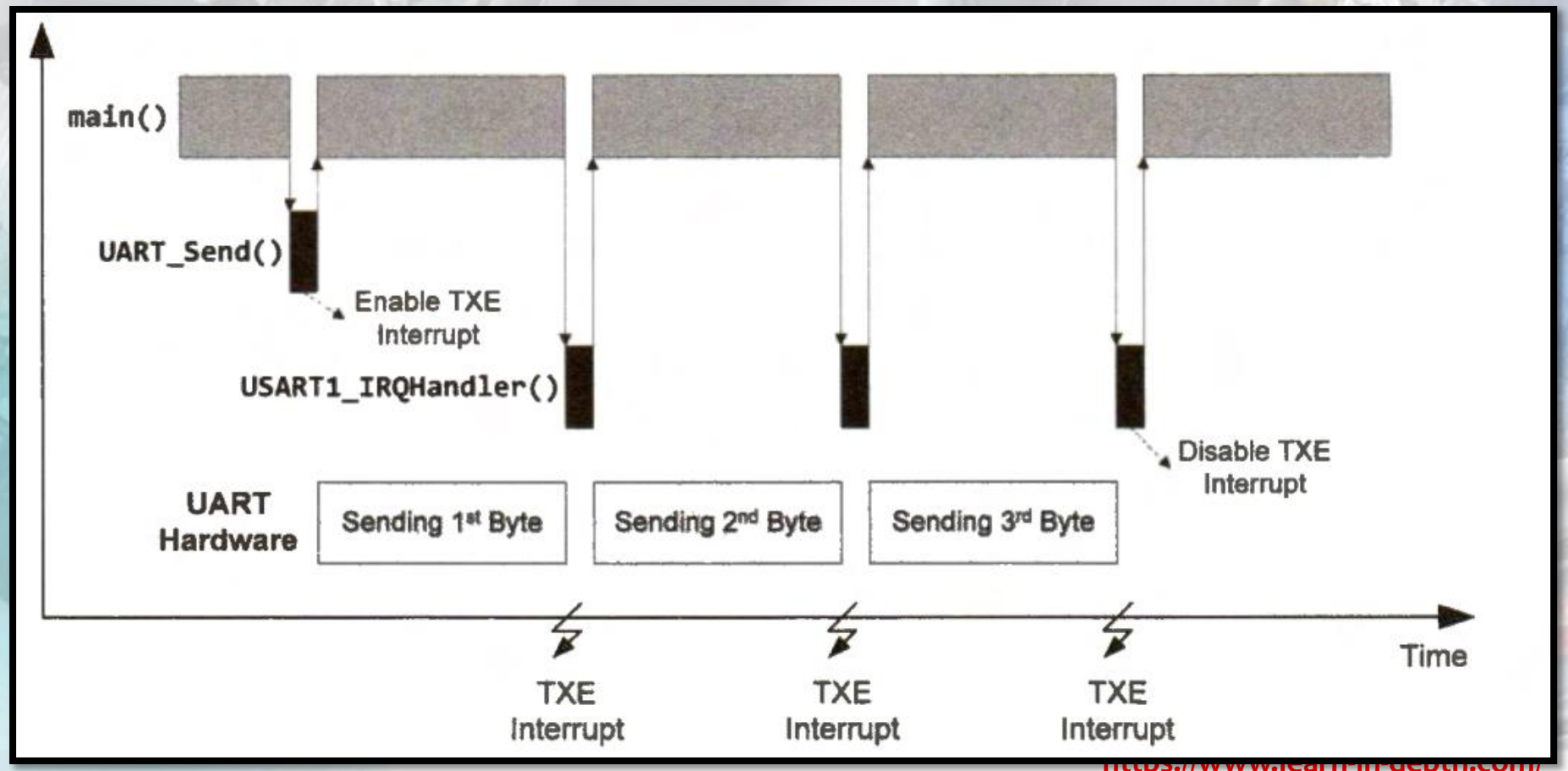
- An USART interrupt can be generated upon the occurrence of several events, such as transmission data register empty (TxE), transmission complete (TC), received data register not empty (RXNE), overrun error detected (ORE), idle line detected (IDLE), and parity error (PE)

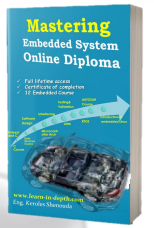


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Sending three bytes via USARTI by using interrupts

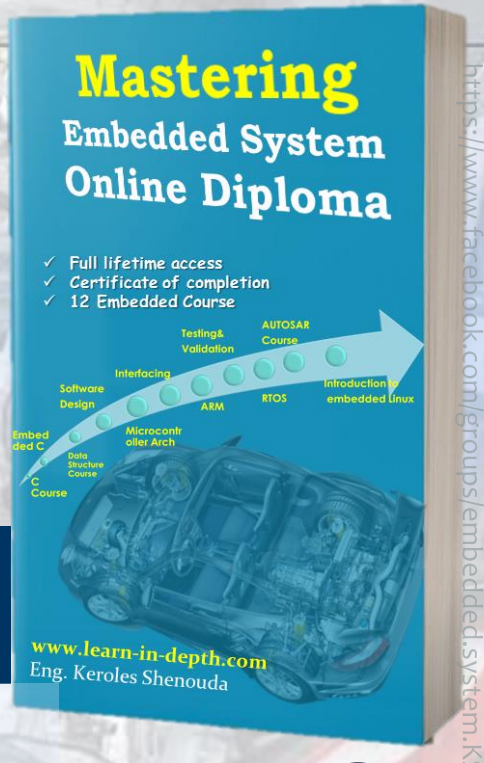




#LEARN_IN DEPTH

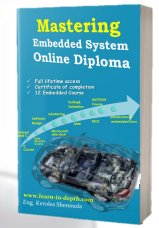
#Be_professional_in_embedded_system

UART Communication via DMA



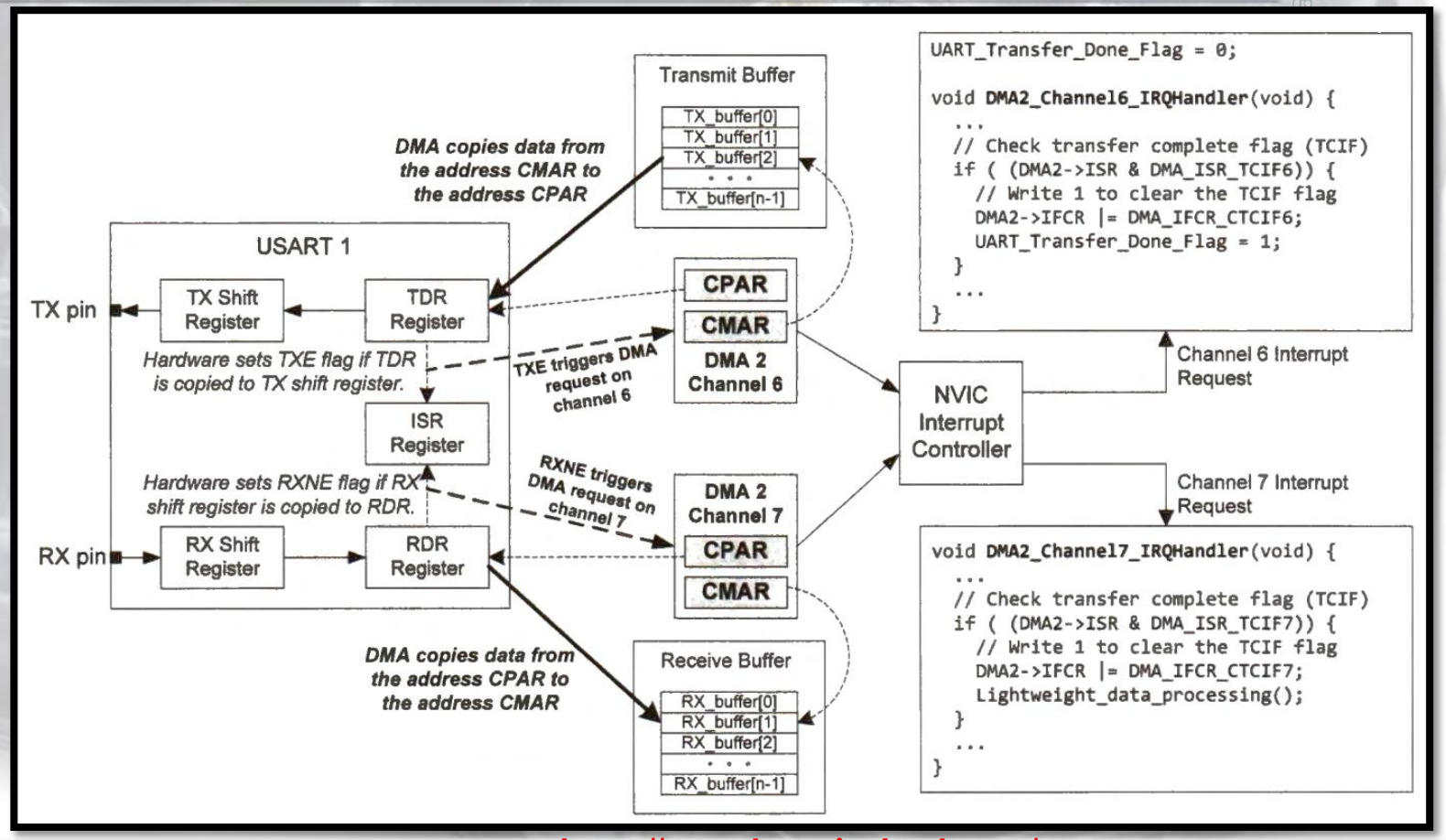
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

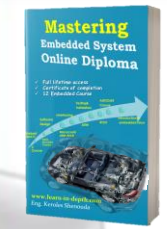


UART Communication via DMA

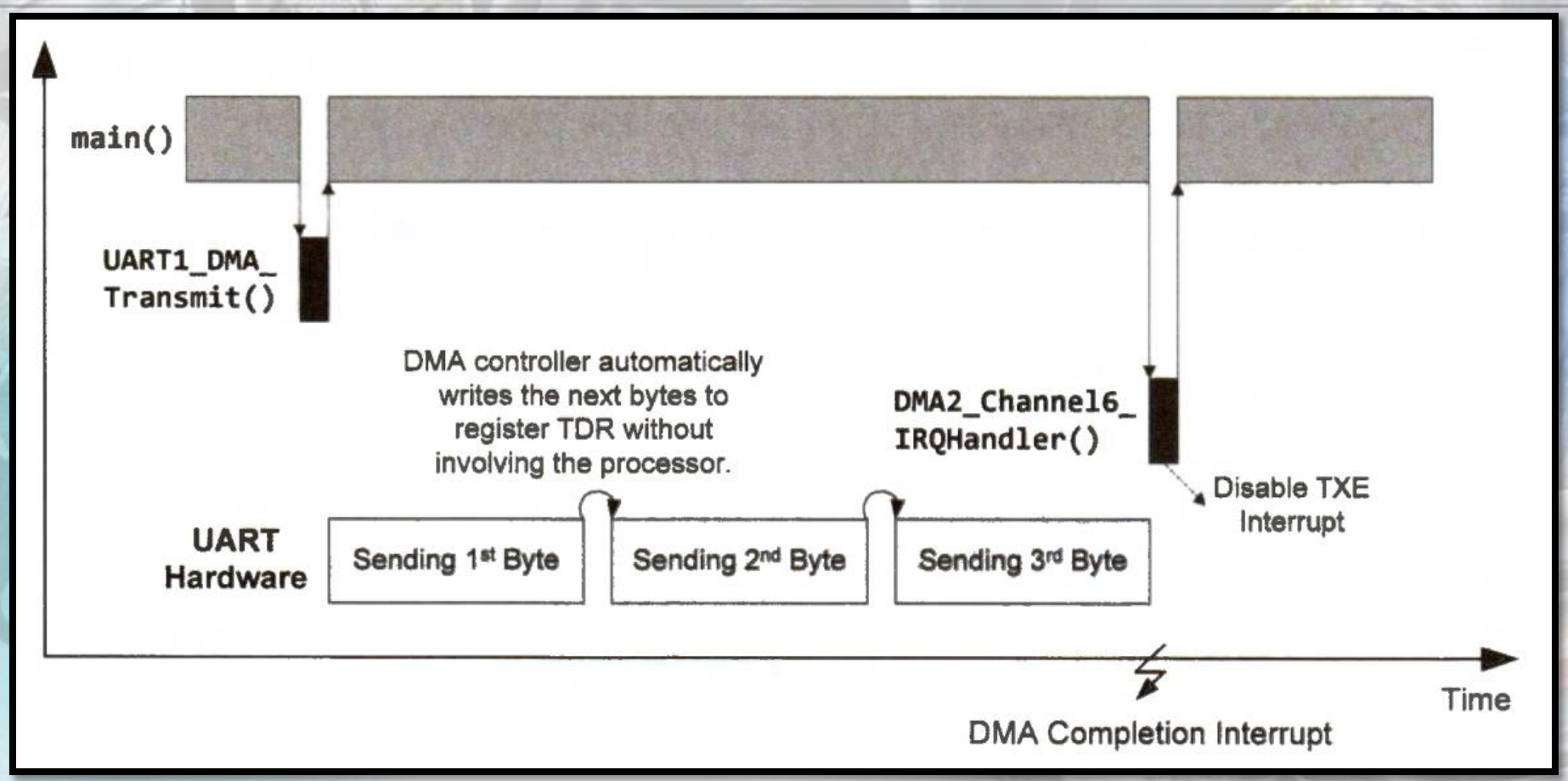
- Using a direct memory access (OMA) controller to move data between a buffer and UART data registers is the most efficient way to perform UART communication
- USARTl_TX and USARTl_RX can be connected to channels 6 and 7 of OMA controller 2, respectively.
- A TXE or RXNE event triggers a OMA request on channel 6 and channel 7, respectively.
- Whenever the TXE bit is set in the ISR register, OMA controller 2 transfers one byte via channel 6 from the memory buffer (pointed to by the CMAR register of OMA 2 channel 6) to the *transmit data register* TDR (pointed to by the CPAR register of OMA 2 channel 6).
- Similarly, whenever the RXNE bit is set in the ISR register, OMA controller 2 transfers one byte via channel 7 from the *receive data register* RDR to the buffer.

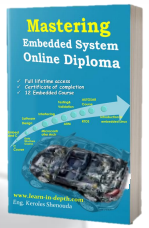


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Sending three bytes via DMA

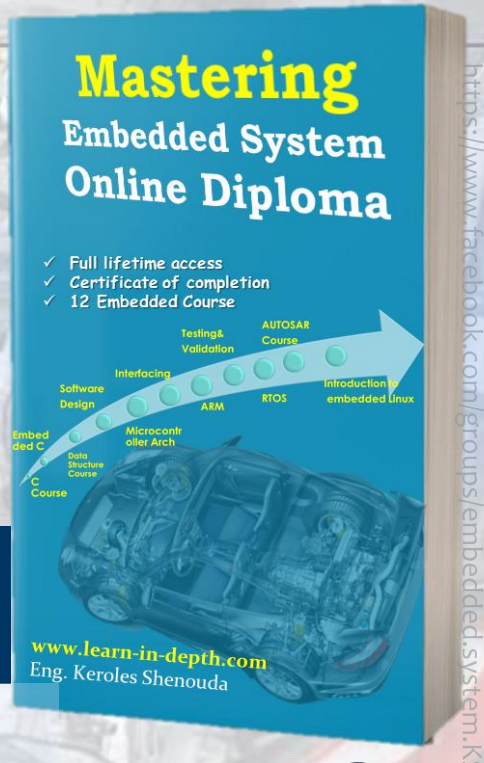




#LEARN_IN DEPTH

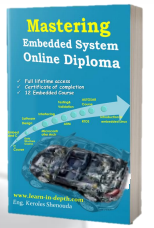
#Be_professional_in_embedded_system

How to read USART Chapter from TRM



LEARN-IN-DEPTH
Be professional in embedded system

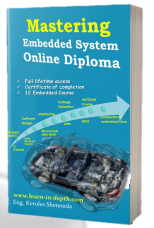
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



USART main features

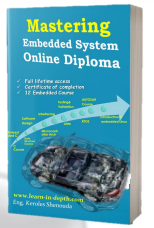
- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- Fractional baud rate generator systems
  - A common programmable transmit and receive baud rates up to 4.5 MBits/s
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
  - 13-bit break generation and 10/11 bit break detection when USART is hardware configured for LIN
- Transmitter clock output for synchronous transmission
- IrDA SIR Encoder Decoder
  - Support for 3/16 bit duration for normal mode
- Smartcard Emulation Capability
  - The Smartcard interface supports the asynchronous protocol Smartcards as defined in ISO 7816-3 standards
  - 0.5, 1.5 Stop Bits for Smartcard operation
- Single wire half duplex communication
- Configurable multibuffer communication using DMA (direct memory access)
  - Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for Transmitter and Receiver

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



USART main features

- Transfer detection flags:
  - Receive buffer full
  - Transmit buffer empty
  - End of Transmission flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- Four error detection flags:
  - Overrun error
  - Noise error
  - Frame error
  - Parity error
- Ten interrupt sources with flags:
  - CTS changes
  - LIN break detection
  - Transmit data register empty
  - Transmission complete
  - Receive data register full
  - Idle line received
  - Overrun error
  - Framing error
  - Noise error
  - Parity error



USART (Read From Specs)

- 27 Universal synchronous asynchronous receiver transmitter (USART)
  - 27.1 USART introduction
  - 27.2 USART main features
  - 27.3 USART functional description
  - 27.4 USART interrupts
  - 27.5 USART mode configuration
  - 27.6 USART registers

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

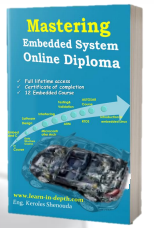
STM32103XX USART

27.3 USART functional description

The interface is externally connected to another device by three pins (see *Figure 279*). Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

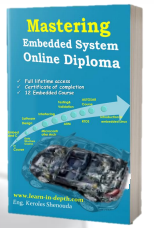
TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its IO port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this IO is used to transmit and receive the data (at USART level, data are then received on SW_RX).



STM32103XX USART

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART_GTPR) in case of Smartcard mode.



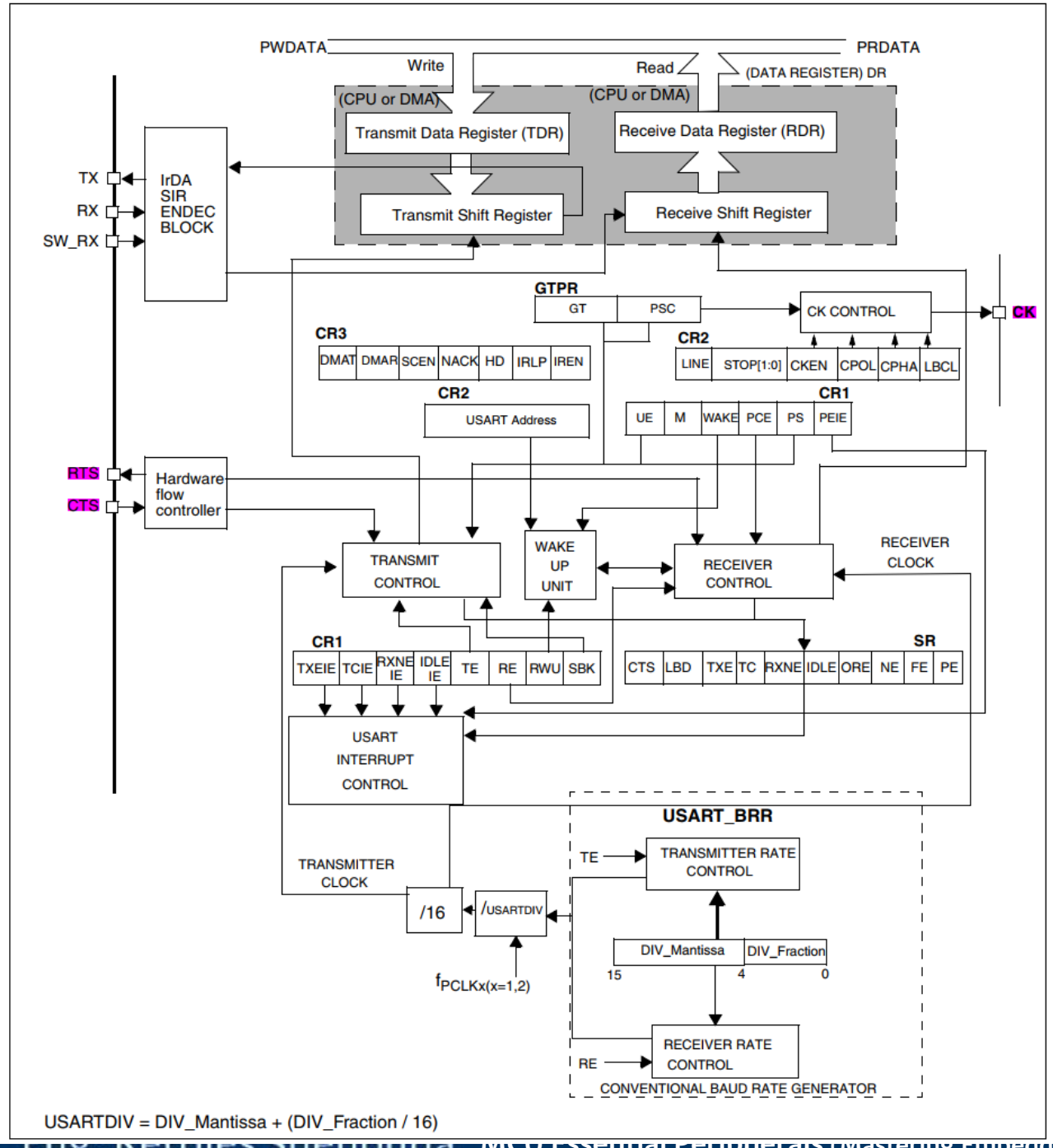
STM32103XX USART

The following pin is required to interface in synchronous mode:

- **CK:** Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In Smartcard mode, CK can provide the clock to the smartcard.

The following pins are required in Hardware flow control mode:

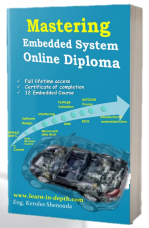
- **CTS:** Clear To Send blocks the data transmission at the end of the current transfer when high
- **RTS:** Request to send indicates that the USART is ready to receive a data (when low).



USARTDIV = DIV_Mantissa + (DIV_Fraction / 16)

- 27 Universal synchronous asynchronous receiver transmitter (USART)
  - 27.1 USART introduction
  - 27.2 USART main features
  - 27.3 USART functional description
    - 27.3.1 USART character description
    - 27.3.2 Transmitter
    - 27.3.3 Receiver
    - 27.3.4 Fractional baud rate generation
    - 27.3.5 USART receiver's tolerance to clock deviation
    - 27.3.6 Multiprocessor communication
    - 27.3.7 Parity control
    - 27.3.8 LIN (local interconnection network) mode
    - 27.3.9 USART synchronous mode
    - 27.3.10 Single-wire half-duplex communication
    - 27.3.11 Smartcard
    - 27.3.12 IrDA SIR ENDEC block
    - 27.3.13 Continuous communication using DMA
    - 27.3.14 Hardware flow control
  - 27.4 USART interrupts
  - 27.5 USART mode configuration
  - 27.6 USART registers

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



STM32103XX USART

27.3.2 Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

Character transmission

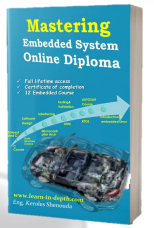
During a USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see *Figure 279*).

Every character is preceded by a start bit, which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

Note: *The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost.*

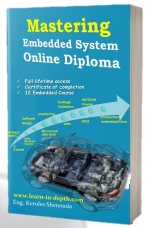
An idle frame will be sent after the TE bit is enabled.



STM32103XX USART

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Select the desired baud rate using the USART_BRR register.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_DR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.



STM32103XX USART

Single byte communication

The TXE bit is always cleared by a write to the data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

STM32103XX USART

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

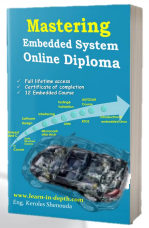
If a frame is transmitted (after the stop bit) and the TXE bit is set, the TC bit goes high. An interrupt is generated if the TCIE bit is set in the USART_CR1 register.

After writing the last data into the USART_DR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low-power mode (see *Figure 282*).

The TC bit is cleared by the following software sequence:

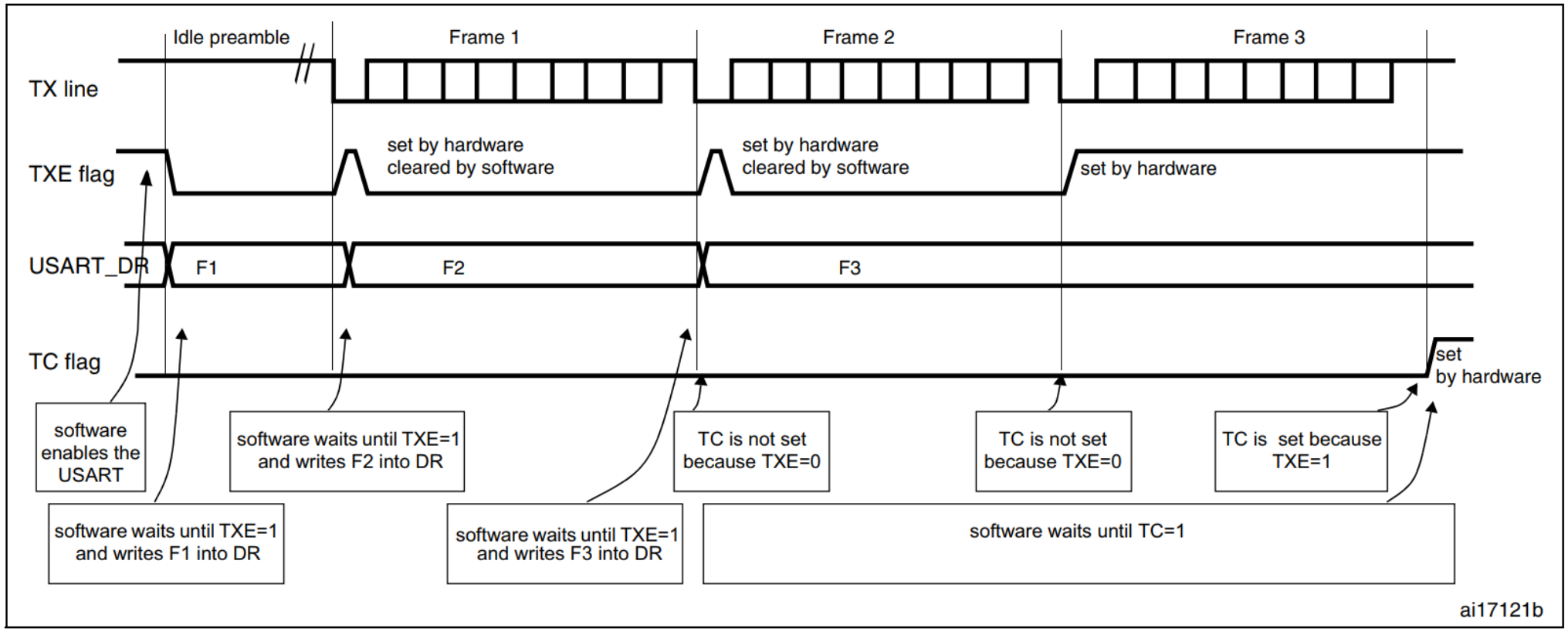
1. A read from the USART_SR register
2. A write to the USART_DR register

Note: The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for Multibuffer communication.

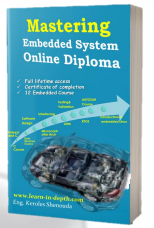


STM32103XX USART

Figure 282. TC/TXE behavior when transmitting



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



STM32103XX USART

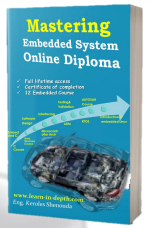
Character reception

During a USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received



STM32103XX USART

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

27.3.4 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

legend: f_{CK} - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

USARTDIV = DIV_Mantissa + (DIV_Fraction / 16)

27.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000

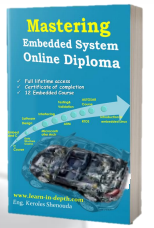
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



STM32103XX USART

Example 2:

To program USARTDIV = 0d25.62

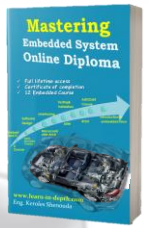
This leads to:

DIV_Fraction = 16*0d0.62 = 0d9.92

The nearest real number is 0d10 = 0xA

DIV_Mantissa = mantissa (0d25.620) = 0d25 = 0x19

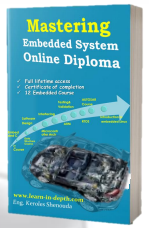
Then, USART_BRR = 0x19A hence USARTDIV = 0d25.625



STM32103XX USART

Baud rate		f_{PCLK} = 36 MHz			f_{PCLK} = 72 MHz		
S.No	in Kbps	Actual	Value programmed in the Baud Rate register	% Error(1)	Actual	Value programmed in the Baud Rate register	% Error(1)
1.	2.4	2.400	937.5	0%	2.4	1875	0%
2.	9.6	9.600	234.375	0%	9.6	468.75	0%
3.	19.2	19.2	117.1875	0%	19.2	234.375	0%
4.	57.6	57.6	39.0625	0%	57.6	78.125	0.%
5.	115.2	115.384	19.5	0.15%	115.2	39.0625	0%
6.	230.4	230.769	9.75	0.16%	230.769	19.5	0.16%
7.	460.8	461.538	4.875	0.16%	461.538	9.75	0.16%
8.	921.6	923.076	2.4375	0.16%	923.076	4.875	0.16%
9.	2250	2250	1	0%	2250	2	0%
10.	4500	NA	NA	NA	4500	1	0%

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



STM32103XX USART

27.4 USART interrupts

Table 196. USART interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission complete	TC	TCIE
Received data ready to be read	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
Break flag	LBD	LBDIE
Noise flag, Overrun error and Framing error in multibuffer communication	NE or ORE or FE	EIE(1)

STM32103XX USART

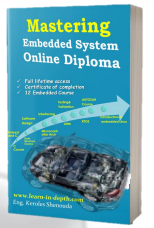
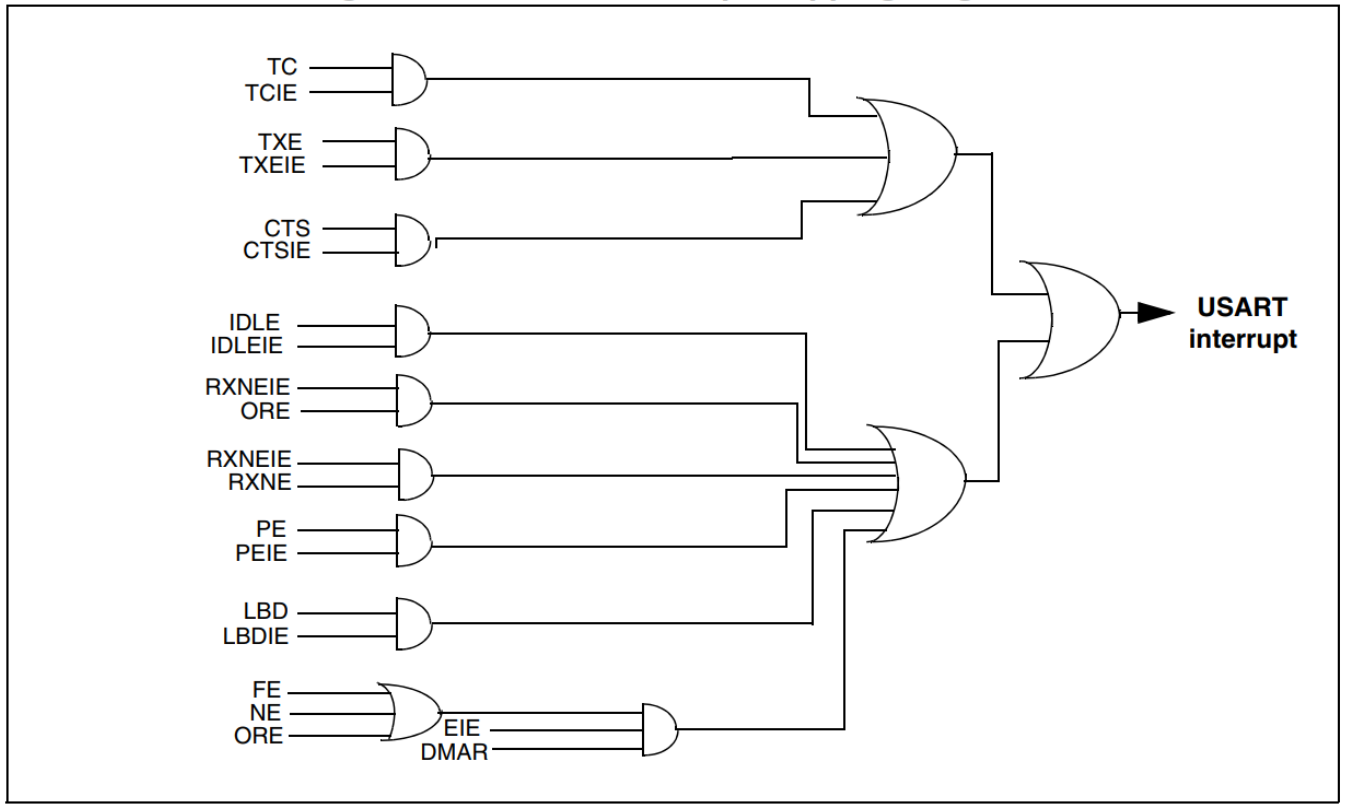
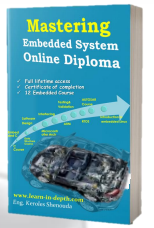


Figure 302. USART interrupt mapping diagram



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



STM32103XX USART

37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC

STM32103XX USART

27.5 USART mode configuration

Table 197. USART mode configuration(1)

USART modes	USART1	USART2	USART3	UART4	UART5
Asynchronous mode	X	X	X	X	X
Hardware Flow Control	X	X	X	NA	NA
Multibuffer Communication (DMA)	X	X	X	X	NA
Multiprocessor Communication	X	X	X	X	X
Synchronous	X	X	X	NA	NA
Smartcard	X	X	X	NA	NA
Half-Duplex (Single-Wire mode)	X	X	X	X	X
IrDA	X	X	X	X	X
LIN	X	X	X	X	X

1. X = supported; NA = not applicable.

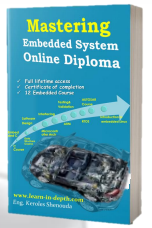
STM32103XX USART

27.6.8 USART register map

The table below gives the USART register map and reset values.

Table 198. USART register map and reset values

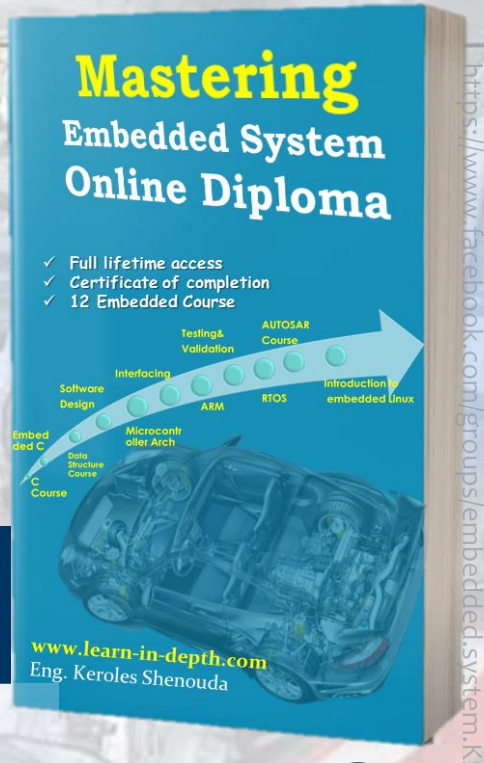
Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																						DR[8:0]									
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	Reserved																						CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																							0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																		GT[7:0]										PSC[7:0]			
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0



#LEARN_IN DEPTH

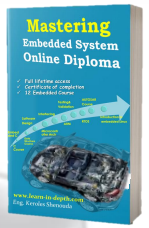
#Be_professional_in_embedded_system

UART Driver



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Application Layer

Application

Keypad Driver .c,.h

ECU Abstraction Layer EAL

LCD Driver .c,.h

MCU.h (MCU Device Header)

APIs
GPIO Driver.c

APIs
EXTI Driver

APIs
USART Driver

Microcontroller Abstraction Layer MCAL

Bare Metal Software (for Target)

MCAL_UART_Init

MCAL_UART_DeInit

MCAL_UART_GPIO_Set_Pins

MCAL_UART_SendData

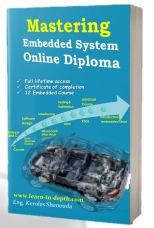
MCAL_UART_ReceiveData

MCAL_UART_WAIT_TC

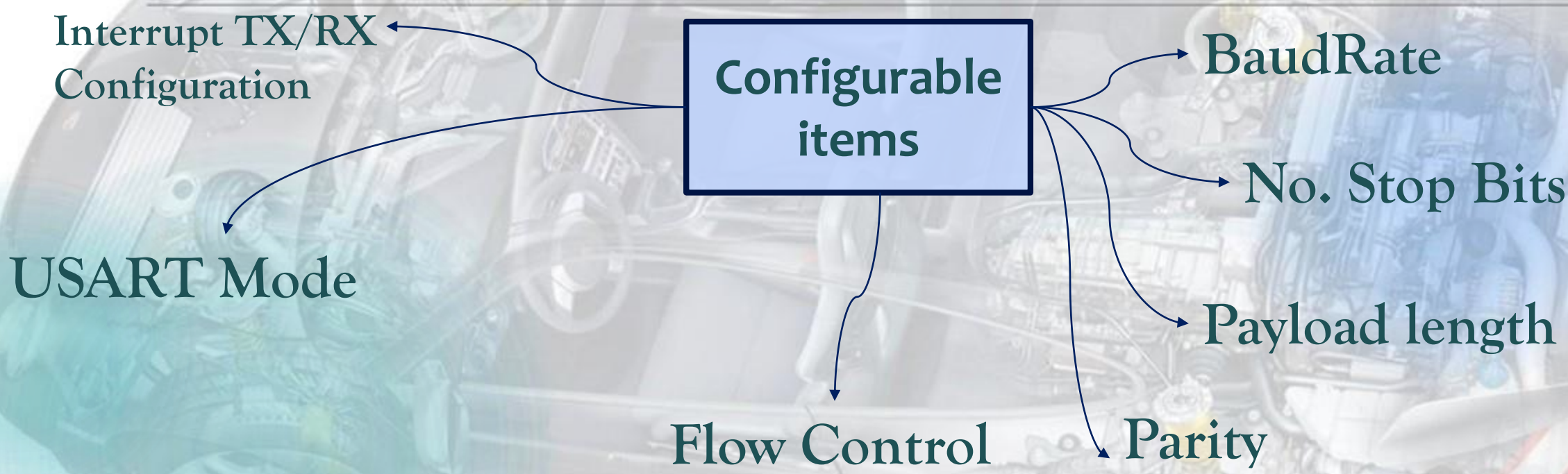
Configurable items

APIs USART Driver.c

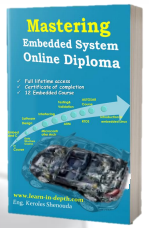
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Configurable items



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

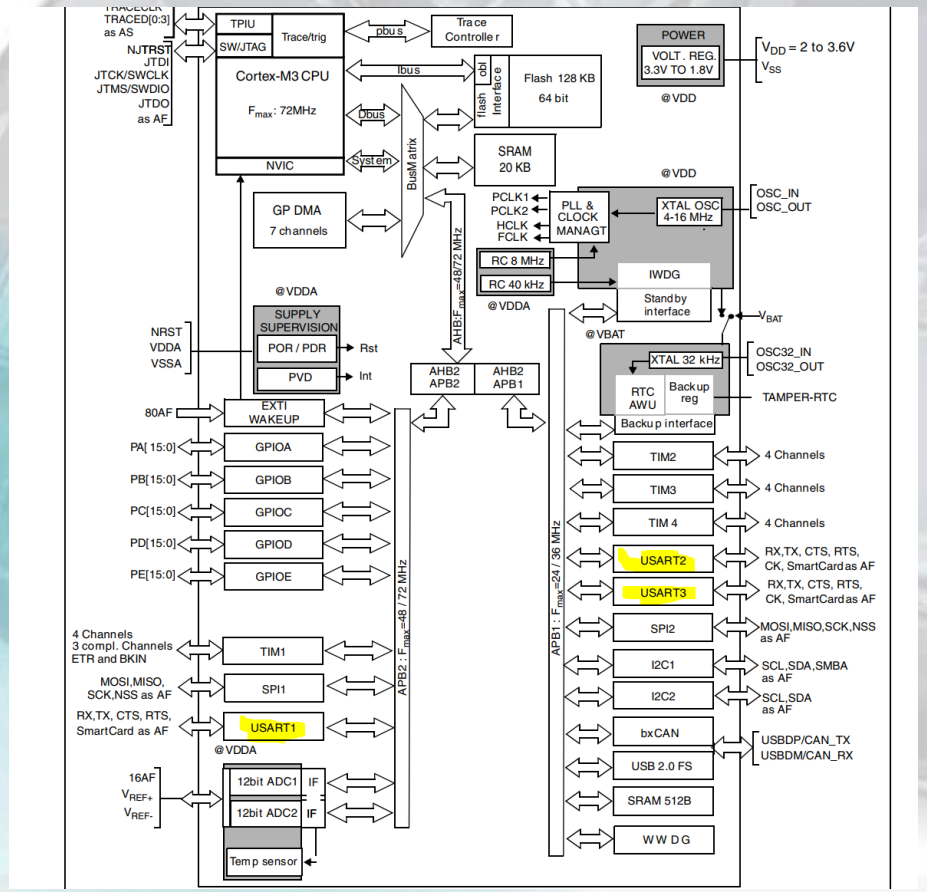


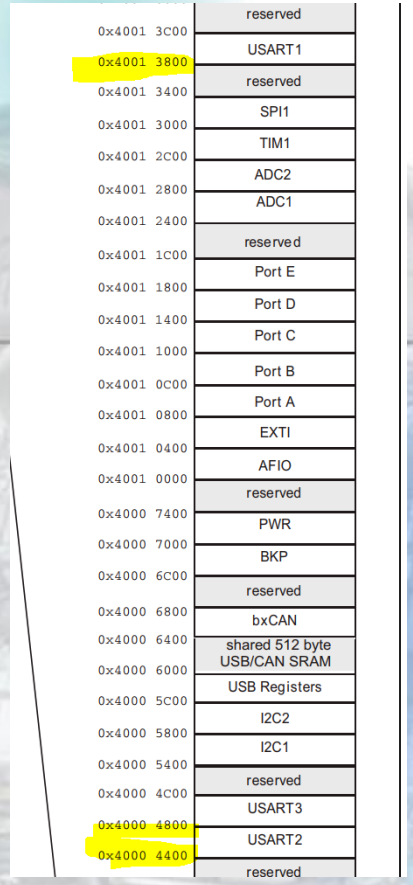
How to write UART Driver

- Figure out Configurable parameters
- Add USART registers on device header
- Add Configuration Structure
- BaudRate: Clock Code Calculations
- Partial Implementation on RCC Driver
- IRQ Handling Support
- APIs Implementation

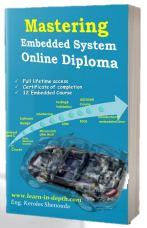
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

1-add USART registers on device header





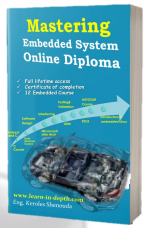

```
61 #define USART1_BASE             (Peripherals_BASE + 0x00013800UL)
62 //------------------------------
63 //Base addresses for APB1 Peripherals
64 //------------------------------
65 #define USART2_BASE             (Peripherals_BASE + 0x00004400UL)
66 #define USART3_BASE             (Peripherals_BASE + 0x00004800UL)
67
```

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



1-add USART registers on device header Cont.

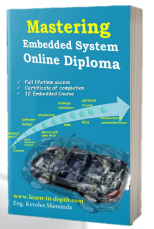
Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	Reserved																								DR[8:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x08	USART_BRR	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	USART_CR3	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	Reserved																GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
143 //-*-*-*-*-*-*-*-*-*-*-*-*-
144 //Peripheral register: USART
145 //-*-*-*-*-*-*-*-*-*-*-*-*
146 typedef struct
147 {
148   volatile uint32_t SR;
149   volatile uint32_t DR;
150   volatile uint32_t BRR;
151   volatile uint32_t CR1;
152   volatile uint32_t CR2;
153   volatile uint32_t CR3;
154   volatile uint32_t GTPR;
155 } USART_TypeDef;
156 //-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

1-add USART registers on device header Cont.

```c
//-*-*-*-*-*-*-*-*-*-*-*-*
//Peripheral Instants:
//-*-*-*-*-*-*-*-*-*-*-*-*

#define GPIOA                    ((GPIO_TypeDef *)GPIOA_BASE)
#define GPIOB                    ((GPIO_TypeDef *)GPIOB_BASE)
#define GPIOC                    ((GPIO_TypeDef *)GPIOC_BASE)
#define GPIOD                    ((GPIO_TypeDef *)GPIOD_BASE)
#define GPIOE                    ((GPIO_TypeDef *)GPIOE_BASE)


#define RCC                      ((RCC_TypeDef *)RCC_BASE)

#define AFIO                     ((AFIO_TypeDef *)AFIO_BASE)
#define EXTI                     ((EXTI_TypeDef *)EXTI_BASE)

#define USART1                     ((USART_TypeDef *)USART1_BASE)
#define USART2                     ((USART_TypeDef *)USART2_BASE)
#define USART3                     ((USART_TypeDef *)USART3_BASE)
```



1-add USART registers on device header Cont.

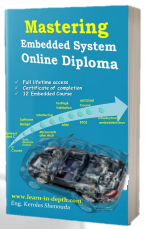
```
202
203 //-*-*-*-*-*-*-*-*-*-*-*-
204 //IVT
205 //-*-*-*-*-*-*-*-*-*-*-*-*
206 //EXTI
207 #define      EXTI0_IRQ           6
208 #define      EXTI1_IRQ           7
209 #define      EXTI2_IRQ           8
210 #define      EXTI3_IRQ           9
211 #define      EXTI4_IRQ           10
212 #define      EXTI5_IRQ           23
213 #define      EXTI6_IRQ           23
214 #define      EXTI7_IRQ           23
215 #define      EXTI8_IRQ           23
216 #define      EXTI9_IRQ           23
217 #define      EXTI10_IRQ          40
218 #define      EXTI11_IRQ          40
219 #define      EXTI12_IRQ          40
220 #define      EXTI13_IRQ          40
221 #define      EXTI14_IRQ          40
222 #define      EXTI15_IRQ          40
223
224 #define      USART1_IRQ          37
225 #define      USART2_IRQ          38
226 #define      USART3_IRQ          39
227
```

Position	Priority	Type of priority	Acronym	Description	Address
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC

```
250 //USART
251 #define NVIC_IRQ37_USART1_Enable    (NVIC_ISER1 |= 1<<( USART1_IRQ - 32 )) //IRQ-32
252 #define NVIC_IRQ38_USART2_Enable    (NVIC_ISER1 |= 1<<( USART2_IRQ - 32 )) //IRQ-32
253 #define NVIC_IRQ39_USART3_Enable    (NVIC_ISER1 |= 1<<( USART3_IRQ - 32 )) //IRQ-32
254
255 #define NVIC_IRQ37_USART1_Disable       (NVIC_ICER1 |= 1<<( USART1_IRQ- 32 )) //IRQ-32
256 #define NVIC_IRQ38_USART2_Disable       (NVIC_ICER1 |= 1<<( USART2_IRQ- 32 )) //IRQ-32
257 #define NVIC_IRQ39_USART3_Disable       (NVIC_ICER1 |= 1<<( USART3_IRQ- 32 )) //IRQ-32
258
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

1-add USART registers on device header Cont.



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC RST	PWR RST	BKP RST	CAN2 RST	CAN1 RST	Reserved		I2C2 RST	I2C1 RST	UART5 RST	UART4 RST	USART3 RST	USART2 RST	Res.
		rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 RST	SPI2 RST	Reserved		WWDG RST	Reserved					TIM7 RST	TIM6 RST	TIM5 RST	TIM4 RST		
rw	rw			rw						rw	rw	rw	rw		

8.3.4 APB2 peripheral reset register (RCC_APB2RSTR)

Address offset: 0x0C

Reset value: 0x00000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1 RST	Res.	SPI1 RST	TIM1 RST	ADC2 RST	ADC1 RST	Reserved		IOPE RST	IOPD RST	IOPC RST	IOPB RST	IOPA RST	Res.	AFIO RST
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

```
206
207 //RCC_Reset
208 #define RCC_USART1_Reset()   ( RCC->APB2RSTR |= (1<<14) )
209 #define RCC_USART2_Reset()   ( RCC->APB1RSTR |= (1<<17) )
210 #define RCC_USART3_Reset()   ( RCC->APB1RSTR |= (1<<18) )
211
212
```


//www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

1-add USART registers on device header Cont.

8.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

8.3.8 APB1 peripheral clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

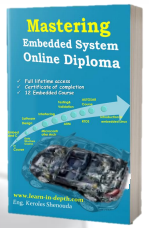
Access: word, half-word and byte access

No wait state, except if the access occurs while an access to a peripheral on APB1 domain is on going. In this case, wait states are inserted until this access to APB1 peripheral is finished.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5EN	UART4EN	USART3 EN	USART2 EN	Res.
		rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

```
201
202 #define RCC_USART1_CLK_EN() ( RCC->APB2ENR |= (1<<14) )
203 #define RCC_USART2_CLK_EN() ( RCC->APB1ENR |= (1<<17) )
204 #define RCC_USART3_CLK_EN() ( RCC->APB1ENR |= (1<<18) )
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

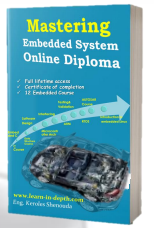


2- add Configuration Structure

```c
 *  Created on: Apr 28, 2021
 *      Author: Keroles Shenouda
 *      Mastering Embedded System Online Diploma
 *      www.learn-in-depth.com
 */

#ifndef STM32F103X8_USART_DRIVER_H_
#define STM32F103X8_USART_DRIVER_H_

//includes
#include "STM32F103x8.h"
#include "stm32f103x8_gpio_driver.h"



//Reference Macros

//@ref USART_define


/*
 * ===============================================================================================
 *                            APIs Supported by "MCAL GPIO DRIVER"
 * ===============================================================================================
 */


#endif /* STM32F103X8_USART_DRIVER_H_ */
```

```c

/*
 * stm32f103x8_USART_driver.h
 *
 *  Created on: Apr 28, 2021
 *      Author: Keroles Shenouda
 *      Mastering Embedded System Online Diploma
 *      www.learn-in-depth.com
 */

#include "stm32f103x8_USART_driver.h"



/*
 * ===============================================================================================
 *                                Generic Variables
 * ===============================================================================================
 */



/*
 * ===============================================================================================
 *                                Generic Macros
 * ===============================================================================================
 */


/*
 * ===============================================================================================
 *                                Generic Functions
 * ===============================================================================================
 */

```

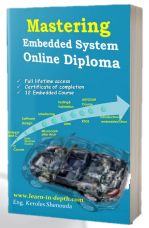
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



2- add Configuration Structure

```
//Configuration structure
typedef struct
{

    uint8_t                    USART_Mode;            // Specifies the TX/RX Mode.
                                                      // This parameter must be set based on @ref USART_Mode_define

    uint32_t                   BaudRate ;             // This member configures the UART communication baud rate
                                                      // This parameter must be set based on @ref BaudRate_define

    uint8_t                    Payload_Length;            // Specifies the number of data bits transmitted or received in a frame.
                                                          // This parameter must be set based on @ref Payload_Length_define


    uint8_t                    Parity ;                   //Specifies the parity mode.
                                                          //@ref Parity_define


    uint8_t                    StopBits ;                 //Specifies the number of stop bits transmitted
                                                          //@ref StopBits_define

    uint8_t                    HwFlowCtl ;                //Specifies whether the hardware flow control mode is enabled or disabled
                                                          //@ref HwFlowCtl_define


    uint8_t                   IRQ_Enable;                 //enable or disable UART IRQ TX/RX
                                                          //@ref IRQ_Enable_define


    void(* P_IRQ_CallBack)(void) ;                    //Set the C Function() which will be called once the IRQ  Happen

}UART_Config;
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



2- add Configuration Structure Cont.

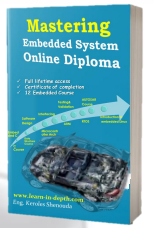
27.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
52// * ==========================================================================
53 //Reference Macros
54 // * ==========================================================================
55
56 //USART_Mode_define
57
58 #define UART_MODE_RX                    (uint32_t) (1<<2)  //RE =1
59 #define UART_MODE_TX                    (uint32_t) (1<<3) //TE =1
60 #define UART_MODE_TX_RX                 ((uint32_t)(1<<2 | 1<<3))
61
```



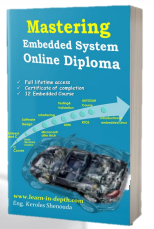
2- add Configuration Structure Cont.

```
62 //UART_BaudRate_define
63
64 #define UART_BaudRate_2400                  2400
65 #define UART_BaudRate_9600                  9600
66 #define UART_BaudRate_19200                 19200
67 #define UART_BaudRate_57600                 57600
68 #define UART_BaudRate_115200                115200
69 #define UART_BaudRate_230400                230400
70 #define UART_BaudRate_460800                460800
71 #define UART_BaudRate_921600                921600
72 #define UART_BaudRate_2250000               2250000
73 #define UART_BaudRate_4500000               4500000
```

Table 192. Error calculation for programmed baud rates

Baud rate		f_{PCLK} = 36 MHz			f_{PCLK} = 72 MHz		
S.No	in Kbps	Actual	Value programmed in the Baud Rate register	% Error(1)	Actual	Value programmed in the Baud Rate register	% Error(1)
1.	2.4	2.400	937.5	0%	2.4	1875	0%
2.	9.6	9.600	234.375	0%	9.6	468.75	0%
3.	19.2	19.2	117.1875	0%	19.2	234.375	0%
4.	57.6	57.6	39.0625	0%	57.6	78.125	0.%
5.	115.2	115.384	19.5	0.15%	115.2	39.0625	0%
6.	230.4	230.769	9.75	0.16%	230.769	19.5	0.16%
7.	460.8	461.538	4.875	0.16%	461.538	9.75	0.16%
8.	921.6	923.076	2.4375	0.16%	923.076	4.875	0.16%
9.	2250	2250	1	0%	2250	2	0%
10.	4500	NA	NA	NA	4500	1	0%

1. Defined as (Calculated Baud Rate - Desired Baud Rate) / Desired Baud Rate.



2- add Configuration Structure Cont.

27.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 12 **M**: Word length

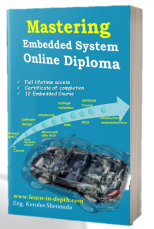
This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, n Stop bit

Note: The M bit must not be modified during a data transfer (both transmission and reception)

```
76 //UART_Payload_Length_define
77
78 #define UART_Payload_Length_8B                (uint32_t)(0)
79 #define UART_Payload_Length_9B                (uint32_t)(1<<12)
80
```



2- add Configuration Structure Cont.

27.6.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

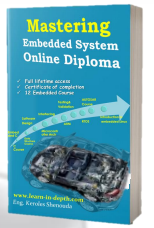
Bit 9 **PS**: Parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity will be selected after the current byte.

0: Even parity

1: Odd parity

```
81 //@ref UART_Parity_define
82
83 #define UART_Parity__NONE          (uint32_t)(0)
84 #define UART_Parity__EVEN          ((uint32_t)1<<10)
85 #define UART_Parity__ODD           ((uint32_t)(1<<10 | 1<<9))
86
```



2- add Configuration Structure Cont.

27.6.5 Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

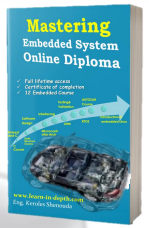
Bits 31:15 Reserved, forced by hardware to 0.

Bit 14 **LINEN**: LIN mode enable
This bit is set and cleared by software.
0: LIN mode disabled
1: LIN mode enabled
The LIN mode enables the capability to send LIN Synch Breaks (13 low bits) using the SBK bit in the USART_CR1 register, and to detect LIN Sync breaks.

Bits 13:12 **STOP**: STOP bits
These bits are used for programming the stop bits.
00: 1 Stop bit
01: 0.5 Stop bit
10: 2 Stop bits
11: 1.5 Stop bit
The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.

```
87 //@ref UART_StopBits_define
88
89 #define UART_StopBits__half              (uint32_t)(1<<12)
90 #define UART_StopBits__1                 (uint32_t)(0)
91 #define UART_StopBits__1_half            (uint32_t)(3<<12)
92 #define UART_StopBits__2                 (uint32_t)(2<<12)
93
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



2- add Configuration Structure Cont.

27.6.6 Control register 3 (USART_CR3)

Address offset: 0x14

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 9 **CTSE**: CTS enable

0: CTS hardware flow control disabled

1: CTS mode enabled, data is only transmitted when the CTS input is asserted (tied to 0). If the CTS input is deasserted while a data is being transmitted, then the transmission is completed before stopping. If a data is written into the data register while CTS is deasserted, the transmission is postponed until CTS is asserted.

This bit is not available for UART4 & UART5.

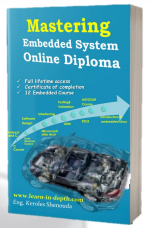
Bit 8 **RTSE**: RTS enable

0: RTS hardware flow control disabled

1: RTS interrupt enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The RTS output is asserted (tied to 0) when a data can be received.

This bit is not available for UART4 & UART5.

```
95 //@ref UART_HwFlowCtl_define
96
97 #define UART_HwFlowCtl_NONE                (uint32_t)(0)
98 #define UART_HwFlowCtl_RTS                 ((uint32_t)1<<8)
99 #define UART_HwFlowCtl_CTS                 ((uint32_t)1<<9)
100 #define UART_HwFlowCtl_RTS_CTS             ((uint32_t)(1<<8 | 1<<9))
101
```



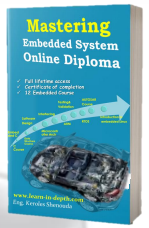
2- add Configuration Structure Cont.

27.4 USART interrupts

Table 196. USART interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission complete	TC	TCIE
Received data ready to be read	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
Break flag	LBD	LBDIE
Noise flag, Overrun error and Framing error in multibuffer communication	NE or ORE or FE	EIE(1)

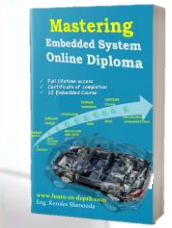
```
102
103 //@ref UART_IRQ_Enable_define
104 #define UART_IRQ_Enable_NONE                    (uint32_t)(0)
105 #define UART_IRQ_Enable_TXE                     (uint32_t) (1<<7) //Transmit data register empty
106 #define UART_IRQ_Enable_TC                      ((uint32_t)(1<<6)) //Transmission complete
107 #define UART_IRQ_Enable_RXNEIE                  (uint32_t) (1<<5) //Received data ready to be read & Overrun error detected
108 #define UART_IRQ_Enable_PE                       (uint32_t) (1<<8) //Parity error
109
110
```




```
17
18  enum UART_IRQ_Event
19  {
20      UART_IRQ_TXE,    //Transmit data register empty
21      UART_IRQ_TC,     //Transmission complete
22      UART_IRQ_RXNE,   //Received data ready to be read
23      UART_IRQ_ORE,    //Overrun error detected
24      UART_IRQ_PE      //Parity error
25  };
26
```

```
void(* P_IRQ_CallBack)(enum UART_IRQ_Event flag) ;                    //Set the C Function() which will be called once the IRQ  Happen
```

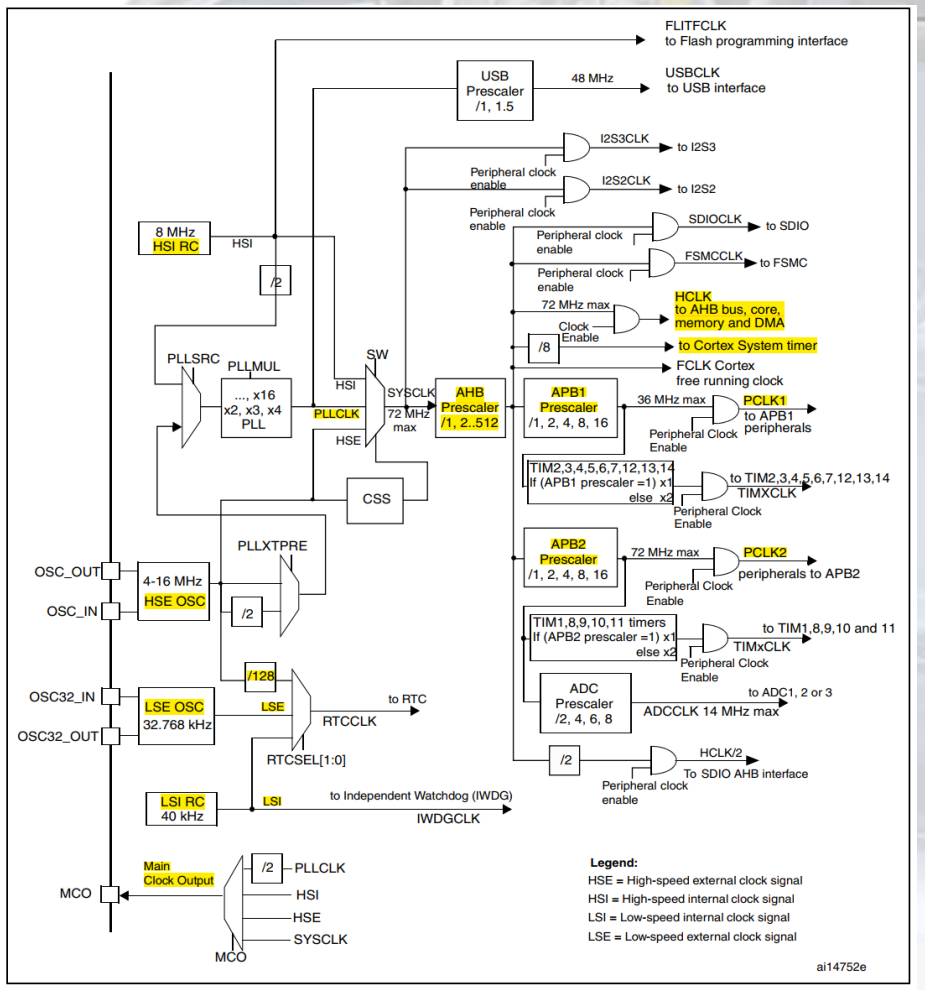
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

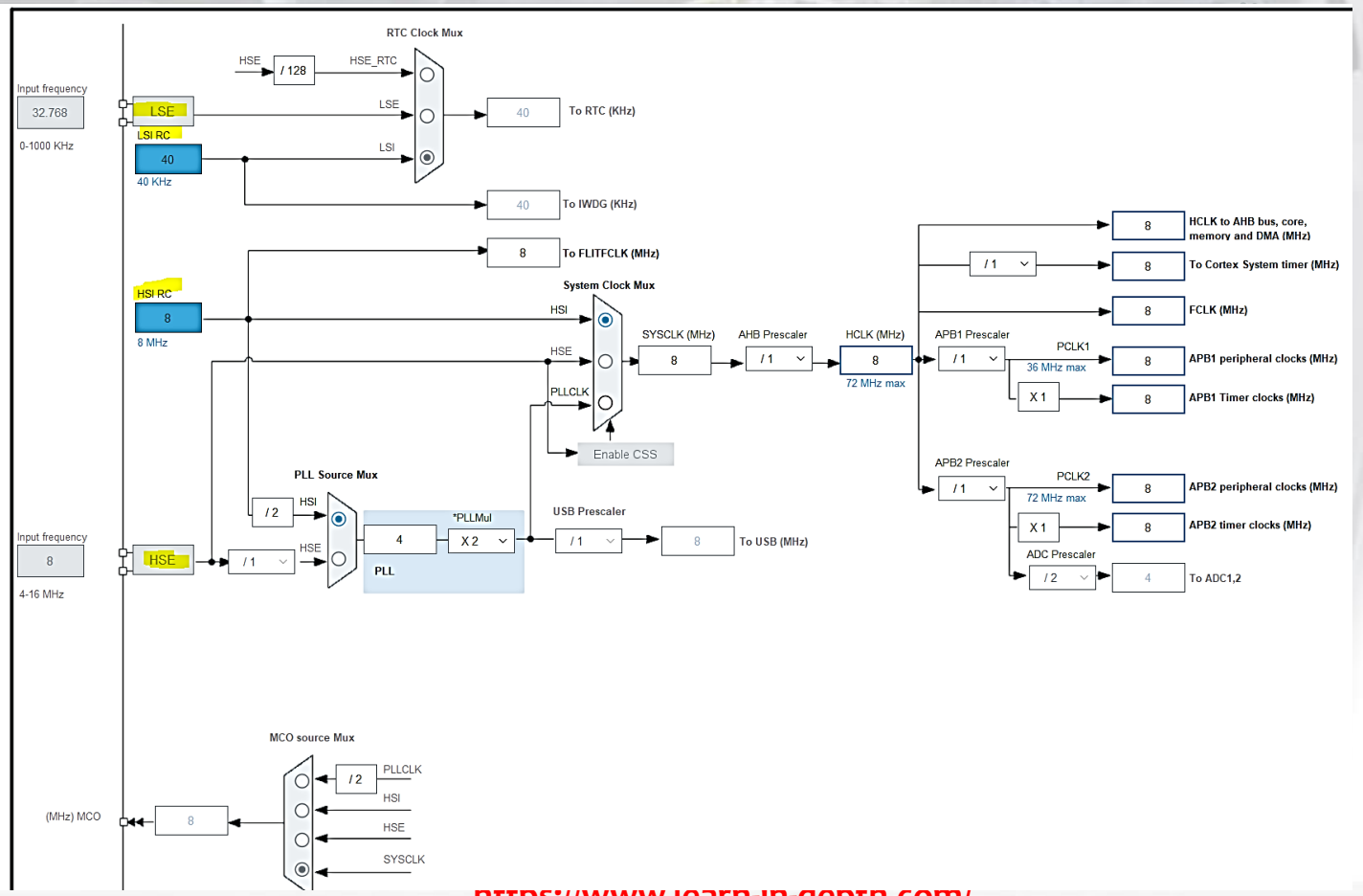


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

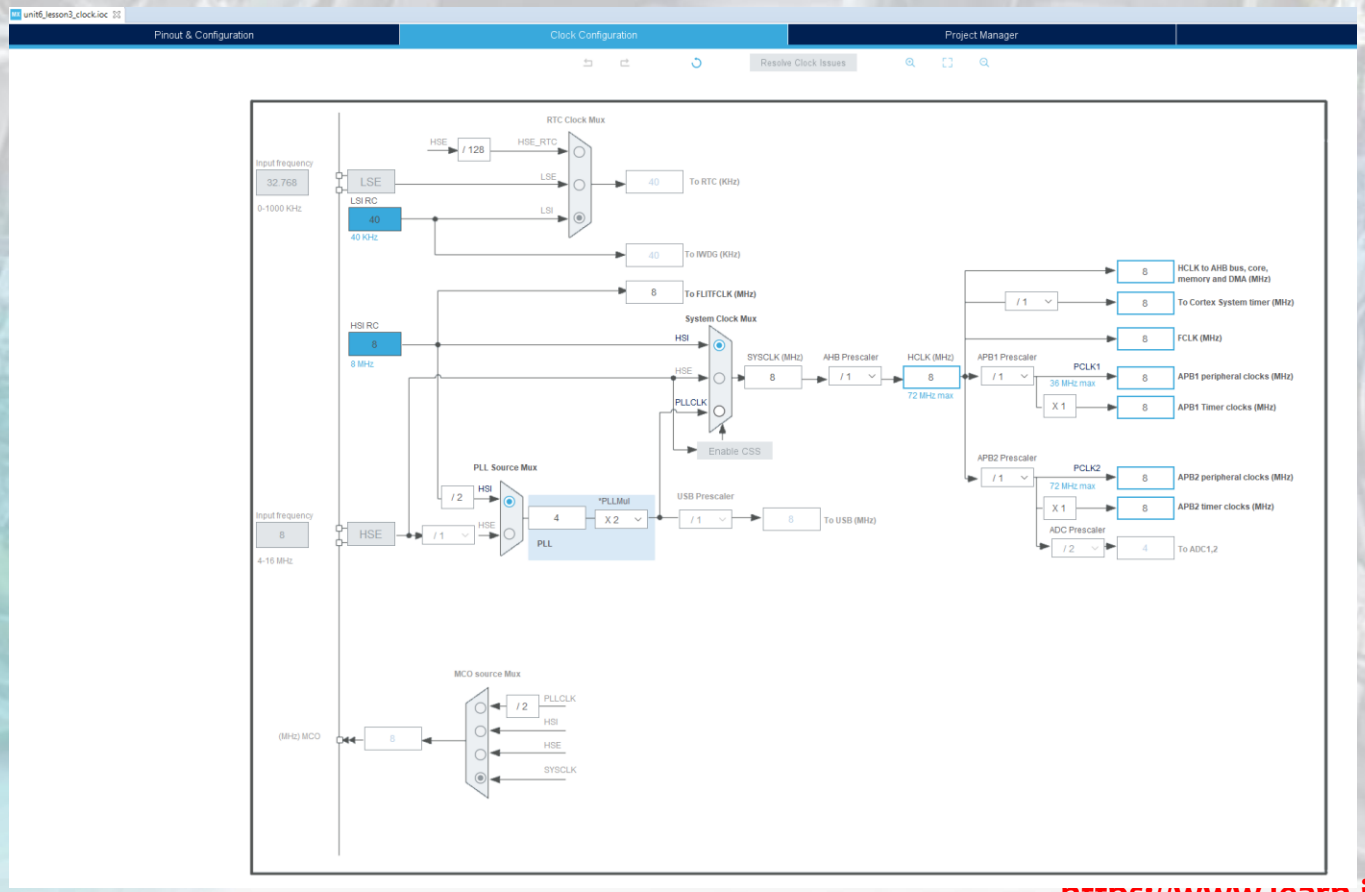
BaudRate: Clock





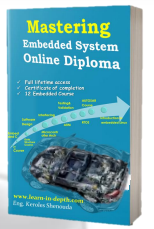
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

BaudRate: Clock



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

BaudRate: Clock

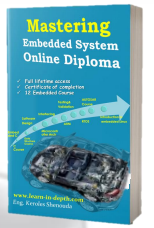


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

```c
uint32_t MCAL_RCC_GetSYS_CLCKFreq(void)
{

//  Bits 3:2 SWS[1:0]: System clock switch status
//  Set and cleared by hardware to indicate which clock source is used as system clock.
//  00: HSI oscillator used as system clock
//  01: HSE oscillator used as system clock
//  10: PLL used as system clock
//  11: Not applicable
    switch  ( (RCC->CFGR  >> 2  ) & 0b11 )
    {
    case 0:

        return HSI_RC_Clk ;
        break ;

    case 1:

        //todo need to calculate  it //HSE User Should Specify it
        return HSE_Clock ;
        break ;

    case 2:

        //todo need to calculate  it PLLCLK and PLLMUL & PLL Source MUX
        return 16000000 ;
        break ;

    }

}
```

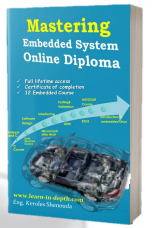

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



BaudRate: Get Clock

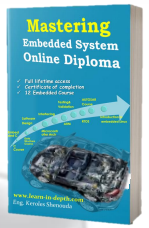
```
90 uint32_t MCAL_RCC_GetHCLKFreq(void)
91 {
92     /* Get HCLK source and Compute PCLK1 frequency ----------------------------*/
93     return (MCAL_RCC_GetSYS_CLCKFreq() >> AHBPrescTable[ ( (RCC->CFGR >> 4 ) & 0xF) ]  ); //the first shift is multiplication}
94 }
95
96 //APB Low speed clock (PCLK1).
97 //Bits 10:8 PPRE1[2:0]: APB Low-speed prescaler (APB1)
98 uint32_t MCAL_RCC_GetPCLK1Freq(void)
99 {
100     /* Get HCLK source and Compute PCLK1 frequency ----------------------------*/
101     return (MCAL_RCC_GetHCLKFreq() >> APBPrescTable[ ( (RCC->CFGR >> 8 ) & 0b111) ]  ); //the first shift is multiplication
102 }
103
104 uint32_t MCAL_RCC_GetPCLK2Freq(void)
105 {
106     /* Get HCLK source and Compute PCLK2 frequency ----------------------------*/
107     return (MCAL_RCC_GetHCLKFreq() >> APBPrescTable[ ( (RCC->CFGR >> 11 ) & 0b111) ]  ); //the first shift is multiplication
108 }
109
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



BaudRate: Get Clock

```
 */

#include "stm32f103x8_RCC_driver.h"

//PPRE1[2:0]: APB Low-speed prescaler (APB1)
//0xx: HCLK not divided
//100: HCLK divided by 2
//101: HCLK divided by 4
//110: HCLK divided by 8
//111: HCLK divided by 16
const uint8_t APBPrescTable[8U] =  {0, 0, 0, 0, 1, 2, 3, 4}; //Shift 1 right == mutliply by 2


//Bits 7:4 HPRE[3:0]: AHB prescaler
//Set and cleared by software to control AHB clock division factor.
//0xxx: SYSCLK not divided
//1000: SYSCLK divided by 2
//1001: SYSCLK divided by 4
//1010: SYSCLK divided by 8
//1011: SYSCLK divided by 16
//1100: SYSCLK divided by 64
//1101: SYSCLK divided by 128
//1110: SYSCLK divided by 256
//1111: SYSCLK divided by 512
const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};

```



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

BaudRate: Get Clock

```
9
10 #ifndef STM32F103X8_RCC_DRIVER_H_
11 #define STM32F103X8_RCC_DRIVER_H_
12
13 //includes
14 #include "STM32F103x8.h"
15
16
17
18 #define  HSE_Clock          (uint32_t)16000000
19 #define  HSI_RC_Clk         (uint32_t)8000000
20
21
22 uint32_t MCAL_RCC_GetSYS_CLCKFreq(void);
23 uint32_t MCAL_RCC_GetHCLKFreq(void);
24 uint32_t MCAL_RCC_GetPCLK1Freq(void);
25 uint32_t MCAL_RCC_GetPCLK2Freq(void);
26
27
28 #endif /* STM32F103X8_RCC_DRIVER_H_ */
29
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

BaudRate: Calculation

27.3.4 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

legend: f_{CK} - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

USARTDIV = DIV_Mantissa + (DIV_Fraction / 16)

- USARTDIV = fclk / (16 * Baudrate)
- DIV_Mantissa = Integer Part (USARTDIV)
- DIV_Fraction = (USARTDIV - DIV_Mantissa) * 16

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

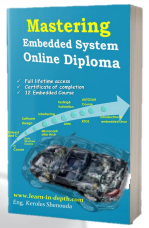
Example

Baud rate		f_{PCLK} = 36 MHz			
S.No	in Kbps	Actual	Value programmed in the Baud Rate register	% Error(1)	Ac
1.	2.4	2.400	937.5	0%	2
2.	9.6	9.600	234.375	0%	9
3.	19.2	19.2	117.1875	0%	19
4.	57.6	57.6	39.0625	0%	57
5.	115.2	115.384	19.5	0.15%	11
6.	230.4	230.769	9.75	0.16%	230
7.	460.8	461.538	4.875	0.16%	461

- USARTDIV = fclk / (16 * Baudrate)
- DIV_Mantissa = Integer Part (USARTDIV)
- DIV_Fraction = (USARTDIV - DIV_Mantissa) * 16

- 36000000 / (115200 * 16) = 19.53 = USARTDIV
- DIV_Mantissa = 19
- DIV_Fraction = 0.53 * 16 = 8.48 The nearest real number is 0d8

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

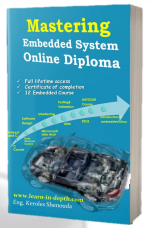


Example The Same But with Coding Trick

- USARTDIV = fclk / (16 * Baudrate)
- **USARTDIV_MUL100** =

  uint32((100 *fclk) / (16 * Baudrate) == (25 *fclk) / (4* Baudrate))
- DIV_Mantissa_MUL100 = Integer Part (USARTDIV) * 100
- DIV_Mantissa = Integer Part (USARTDIV)
- DIV_Fraction = ((USARTDIV_MUL100 - DIV_Mantissa_MUL100) * 16) / 100

- 36000000 / (115200 * 16) = (uint32) 19.53 = 19 = USARTDIV
- 36000000*25 / (115200 * 4) = (uint32) 1953.125 = 1953 = **USARTDIV_MUL100**
- DIV_Mantissa_MUL100 = 19 * 100 = 1900
- DIV_Fraction = ((1953 - 1900) * 16)/100 = 848/100 = 8 (uint32)

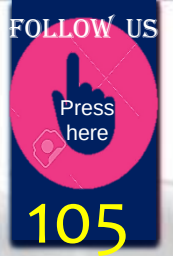


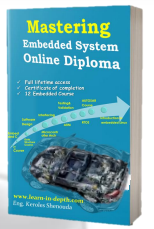
BaudRate Calculation Test

```c
#include "stdio.h"
#include "stdlib.h"
#include <stdint.h>

#define USARTDIV(_PCLK_, _BAUD_)                        (uint32_t) (_PCLK_/(16 * _BAUD_ ))
#define USARTDIV_MUL100(_PCLK_, _BAUD_)                 (uint32_t) ( (25 * _PCLK_ ) / (4  * _BAUD_ ))
#define Mantissa_MUL100(_PCLK_, _BAUD_)                 (uint32_t) (USARTDIV(_PCLK_, _BAUD_) * 100)
#define Mantissa(_PCLK_, _BAUD_)                        (uint32_t) (USARTDIV(_PCLK_, _BAUD_) )
#define DIV_Fraction(_PCLK_, _BAUD_)                    (uint32_t) (((USARTDIV_MUL100(_PCLK_, _BAUD_) -  Mantissa_MUL100(_PCLK_, _BAUD_) ) * 16 ) / 100 )
#define UART_BRR_Register(_PCLK_, _BAUD_)               (( Mantissa (_PCLK_, _BAUD_) ) <<4 )|( (DIV_Fraction(_PCLK_, _BAUD_)) & 0xF )


void main ()
{


    uint32_t DIV = USARTDIV(36000000, 115200) ;
    uint32_t Mant = Mantissa(36000000, 115200) ;
    uint32_t Fraction = DIV_Fraction(36000000, 115200) ;
    uint32_t BRR_Register  = UART_BRR_Register(36000000, 115200) ;


    printf (" DIV=%d  Mant=%d Fraction=%d  BRR_Register=%d   \n", DIV,   Mant,Fraction, BRR_Register);


}
```

```
MINGW64:/d/courses/new_diploma/Diploma online/Second_term/labs
kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/Diploma online/Second_term/labs
$ gcc main.c
main.c: In function 'main':
main.c:14: warning: return type of 'main' is not 'int'
main.c:26:2: warning: no newline at end of file

kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/Diploma online/Second_term/labs
$ ./a
a.exe           Ahmed     labs/ ARM_A53/        Atmega32-labs/

kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/Diploma online/Second_term/labs
$ ./a.exe
 DIV=19  Mant=19 Fraction=8  BRR_Register=312

kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new_diploma/Diploma online/Second_term/labs
$
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/





#LEARN_IN DEPTH

#Be_professional_in_embedded_system

BaudRate Calculation Code

```
125
126 //BaudRate Calculation
127 //USARTDIV = fclk / (16 * Baudrate)
128 //USARTDIV_MUL100 =
129 // uint32((100 *fclk ) / (16 * Baudrate) == (25 *fclk ) / (4* Baudrate) )
130 //DIV_Mantissa_MUL100 = Integer Part (USARTDIV  )  * 100
131 //DIV_Mantissa = Integer Part (USARTDIV  )
132 //DIV_Fraction = (( USARTDIV_MUL100  - DIV_Mantissa_MUL100  ) * 16 ) / 100
133
134 #define USARTDIV(_PCLK_, _BAUD_)                          (uint32_t) (_PCLK_/(16 * _BAUD_ ))
135 #define USARTDIV_MUL100(_PCLK_, _BAUD_)                   (uint32_t) ( (25 * _PCLK_ ) / (4  * _BAUD_ ))
136 #define Mantissa_MUL100(_PCLK_, _BAUD_)                   (uint32_t) (USARTDIV(_PCLK_, _BAUD_) * 100)
137 #define Mantissa(_PCLK_, _BAUD_)                          (uint32_t) (USARTDIV(_PCLK_, _BAUD_) )
138 #define DIV_Fraction(_PCLK_, _BAUD_)                      (uint32_t) (((USARTDIV_MUL100(_PCLK_, _BAUD_) -  Mantissa_MUL100(_PCLK_, _BAUD_) )* 16 )
139 #define UART_BRR_Register(_PCLK_, _BAUD_)                 (( Mantissa (_PCLK_, _BAUD_) ) <<4 )|( (DIV_Fraction(_PCLK_, _BAUD_)) & 0xF )
140
141
142
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

```c
void MCAL_UART_SendData (USART_TypeDef *USARTx, uint16_t *pTxBuffer,enum Polling_mechism PollingEn )
{
    uint16_t *pdata;

    // wait until TXE flag is set in the SR
    if (PollingEn == enable)
        while(! (USARTx->SR & 1<<7 ) );

    //Check the USART_WordLength item for 9BIT or 8BIT in a frame
    if (Global_UART_Config->Payload_Length == UART_Payload_Length_9B)
    {
        //if 9BIT, load the DR with 2bytes masking the bits other than first 9 bits

        //          When transmitting with the parity enabled (PCE bit set to 1 in the USART_C
        //          the value written in the MSB (bit 7 or bit 8 depending on the data length)
        //          because it is replaced by the parity.
        //          When receiving with the parity enabled, the value read in the MSB bit is t
        //          bit.
        //
        USARTx->DR = (*pTxBuffer & (uint16_t)0x01FF);

    }else
    {
        //This is 8bit data transfer
        USARTx->DR = (*pTxBuffer  & (uint8_t)0xFF);
    }


}

```

27.6.2 Data register (USART_DR)

Address offset: 0x04

Reset value: Undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							DR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:9 Reserved, forced by hardware to 0.

Bits 8:0 **DR[8:0]**: Data value

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

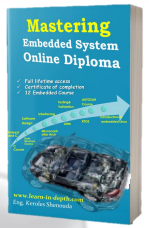
The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting with the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

IRQ USART 1

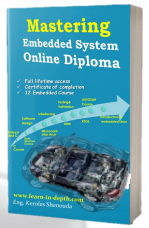
```
234
235 //ISR
236 void USART1_IRQHandler (void)
237 {
238     enum UART_IRQ_Event flag ;
239
240     //Transmit data register empty
241     if (USART1->SR & 1<<7  )
242         flag = UART_IRQ_TXE ;
243
244     //Transmission complete
245     else if (USART1->SR & 1<<6  ){
246         flag = UART_IRQ_TC ;
247         //The TC bit can also be cleared by writing a '0' to it
248         USART1->SR &= ~(1<<6) ;
249     }
250     //Received data ready to be read
251     else if (USART1->SR & 1<<5  ){
252         flag = UART_IRQ_RXNE ;
253         //The RXNE flag can also be cleared bywriting a zero to it
254         USART1->SR &= ~(1<<5 ) ;
255     }
256
257     //Overrun error detected
258     else if (USART1->SR & 1<<3  ){
259         flag = UART_IRQ_ORE ;
260     }
261
262     ////Parity error
263     else if (USART1->SR & 1<<0  ){
264         flag = UART_IRQ_PE ;
265     }
266
267
268     Global_UART_Config->P_IRQ_CallBack (flag) ;
269
270 }
271
```



#LEARN_IN DEPTH

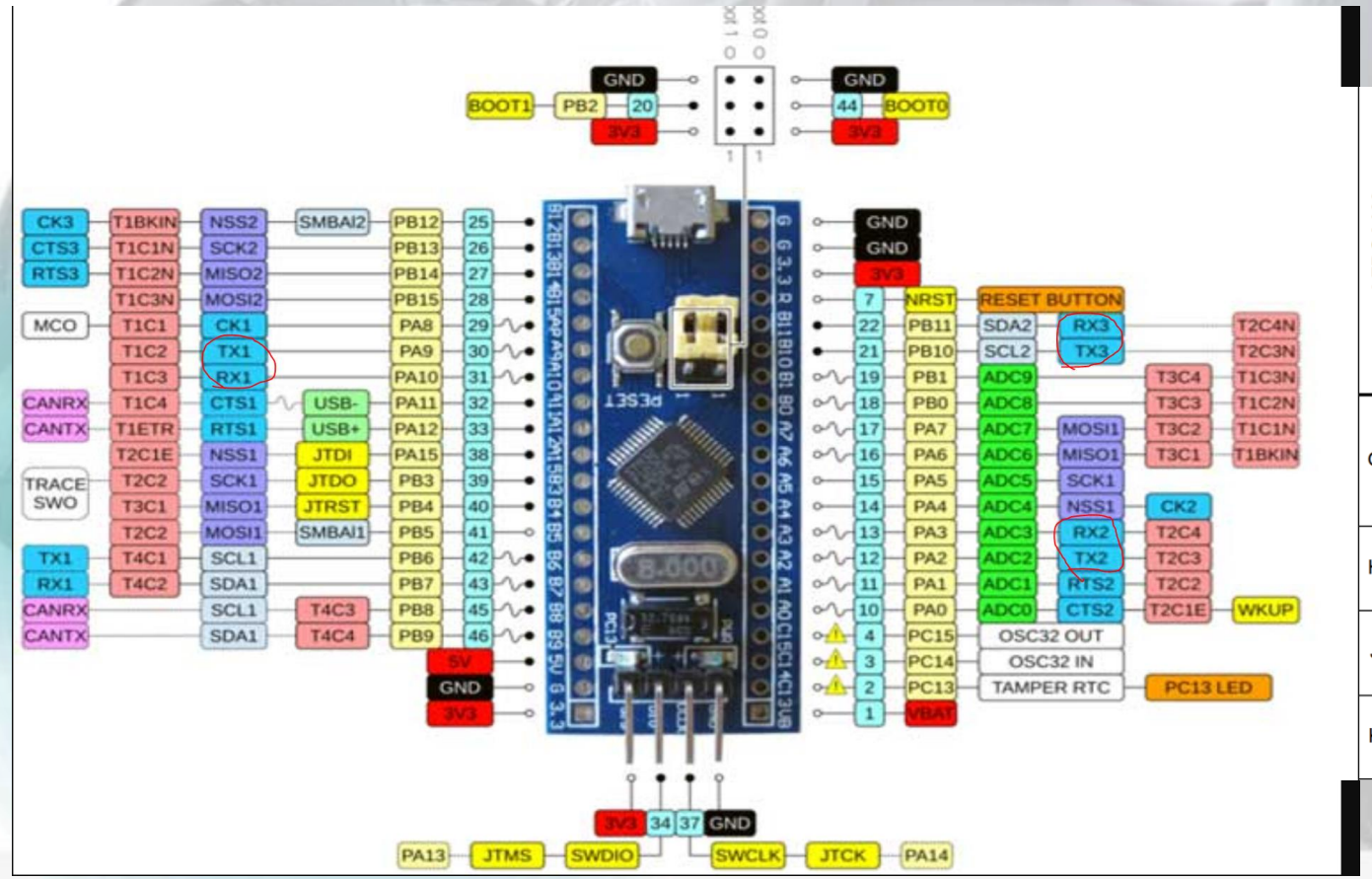
#Be_professional_in_embedded_system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



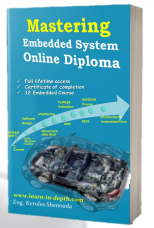

```c
void MCAL_UART_ReceiveData  (USART_TypeDef *USARTx, uint16_t *pRxBuffer ,enum Polling_mechism PollingEn )
{
    //Loop over until "Len" number of bytes are transferred
    //wait until RXNE flag is set in the SR
    if (PollingEn == enable)
    {
        while( ! (USARTx->SR & 1<<5 ));
    }


    //Check the USART_WordLength item for 9BIT or 8BIT in a frame
    if (Global_UART_Config->Payload_Length == UART_Payload_Length_9B)
    {
        if (Global_UART_Config->Parity ==UART_Parity__NONE)
        {
            //no parity So all 9bit are considered data
            *((uint16_t*) pRxBuffer) = USARTx->DR ;

        }else
        {
            //Parity is used, so, 8bits will be of user data and 1 bit is parity
            *((uint16_t*) pRxBuffer) = ( USARTx->DR  & (uint8_t)0xFF );

        }

    }else
    {
        //This is 8bit data
        if (Global_UART_Config->Parity ==UART_Parity__NONE)
        {
            //no parity So all 8bit are considered data
            *((uint16_t*) pRxBuffer) = ( USARTx->DR  & (uint8_t)0xFF ) ;

        }else
        {
            //Parity is used, so,7 bits will be of user data and 1 bit is parity
            *((uint16_t*) pRxBuffer) = ( USARTx->DR  & (uint8_t)0X7F );

        }
    }


}
```

UART PIN



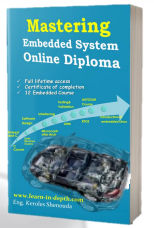
LFBGA100	UFBG100	LQFP48/UFQFPN48	TFBGA64	LQFP64	LQFP100	VFQFPN36	Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Default	Remap
G2	L2	10	G2	14	23	7	PA0-WKUP	I/O	-	PA0	WKUP/ USART2_CTS(9)/ ADC12_IN0/ TIM2_CH1_ ETR(9)	-
H2	M2	11	H2	15	24	8	PA1	I/O	-	PA1	USART2_RTS(9)/ ADC12_IN1/ TIM2_CH2(9)	-
J2	K3	12	F3	16	25	9	PA2	I/O	-	PA2	USART2_TX(9)/ ADC12_IN2/ TIM2_CH3(9)	-
K2	L3	13	G3	17	26	10	PA3	I/O	-	PA3	USART2_RX(9)/ ADC12_IN3/ TIM2_CH4(9)	-

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



J7	L10	21	G7	29	47	-	PB10	I/O	FT	PB10	I2C2_SCL/ USART3_TX(9)
K7	L11	22	H7	30	48	-	PB11	I/O	FT	PB11	I2C2_SDA/ USART3_RX(9)

J8	K12	26	G8	34	52	-	PB13	I/O	FT	PB13	SPI2_SCK/ USART3_CTS(9)/ TIM1_CH1N (9)
H8	K11	27	F8	35	53	-	PB14	I/O	FT	PB14	SPI2_MISO/ USART3_RTS(9) TIM1_CH2N (9)



C9	D10	30	C7	42	68	21	PA9	I/O	FT	PA9	USART1_TX(9)/ TIM1_CH2(9)
D10	C12	31	C6	43	69	22	PA10	I/O	FT	PA10	USART1_RX(9)/ TIM1_CH3(9)
C10	B12	32	C8	44	70	23	PA11	I/O	FT	PA11	USART1_CTS/ CANRX(9)/ USBDM/ TIM1_CH4(9)
B10	A12	33	B8	45	71	24	PA12	I/O	FT	PA12	USART1_RTS/ CANTX(9) /USBDP TIM1_ETR(9)

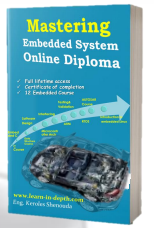


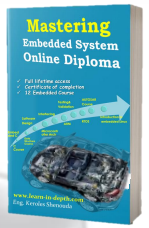
Table 24. USARTs

USART pinout	Configuration	GPIO configuration
USARTx_TX(1)	Full duplex	Alternate function push-pull
	Half duplex synchronous mode	Alternate function push-pull

Table 24. USARTs (continued)

USART pinout	Configuration	GPIO configuration
USARTx_RX	Full duplex	Input floating / Input pull-up
	Half duplex synchronous mode	Not used. Can be used as a general IO
USARTx_CK	Synchronous mode	Alternate function push-pull
USARTx_RTS	Hardware flow control	Alternate function push-pull
USARTx_CTS	Hardware flow control	Input floating/ Input pull-up

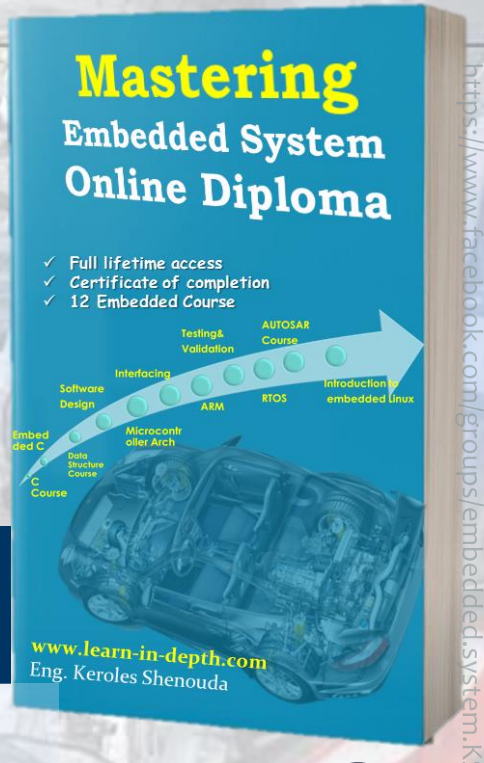
1. The USART_TX pin can also be configured as alternate function open drain.



#LEARN_IN DEPTH

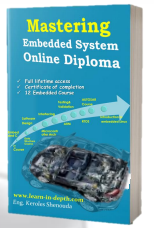
#Be_professional_in_embedded_system

CASE Study 1 with Debugging : using Polling mechanism

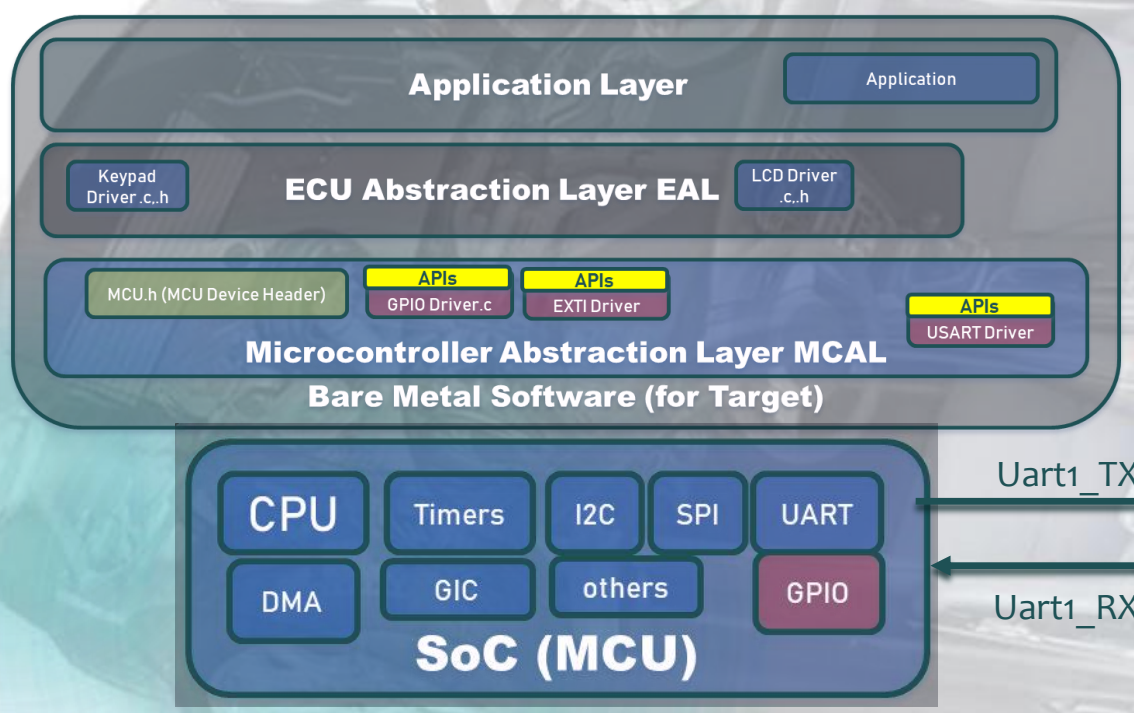


LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

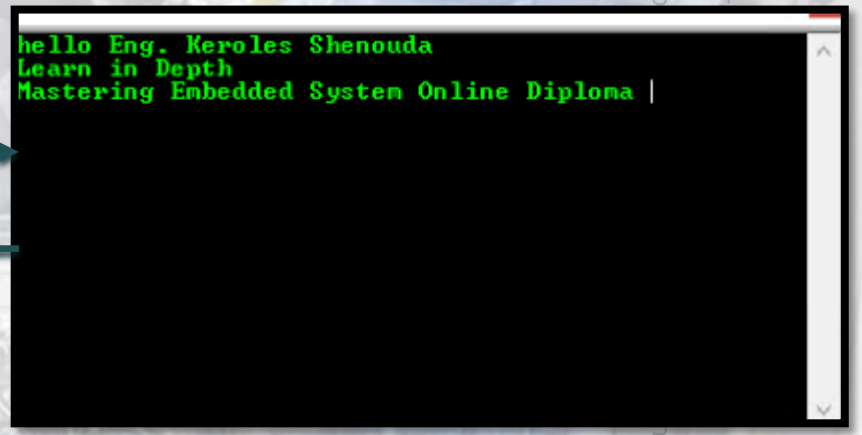


CASE Study 1 with Debugging : using Polling mechanism



Uart1_TX

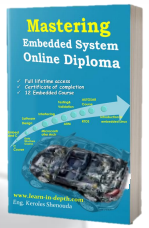
Uart1_RX



Terminal
Baudrate = 115200
Parity = None
Payload_width = 8 bit
Stop = 1
Flow Control = None

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

CASE Study 1 with Debugging : using Polling mechanism

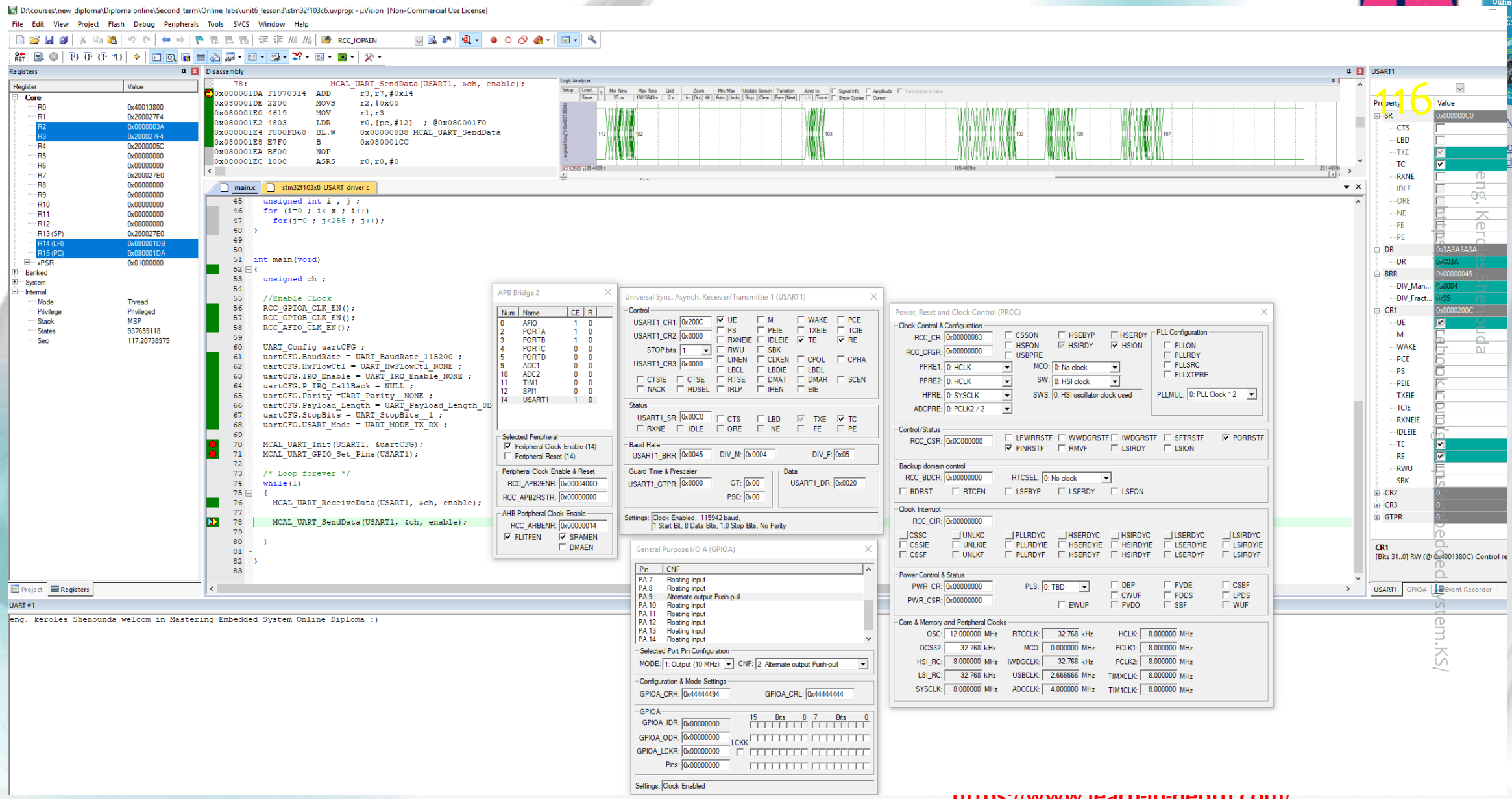


#LEARN_IN DEPTH

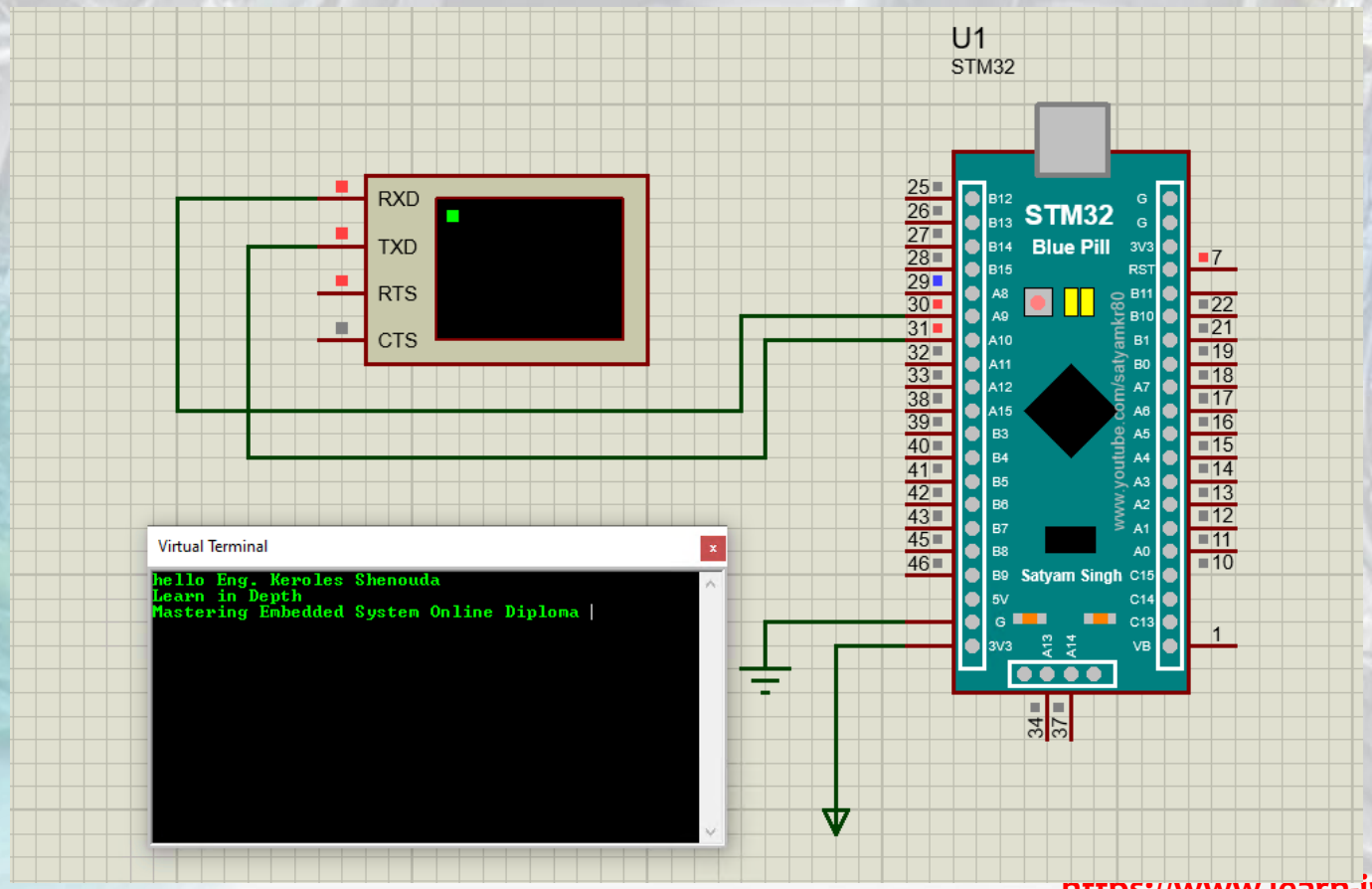
#Be_professional_in_embedded_system


```
//Learn-in-depth
//Keroles Shenouda
//Mastering_Embedded System online diploma
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

#include "STM32F103x8.h"
#include "stm32f103x8_gpio_driver.h"
#include "lcd.h"
#include "keypad.h"
#include "stm32f103x8_EXTI_driver.h"
#include "stm32f103x8_USART_driver.h"


unsigned int IRQ_Flag = 0 ;

void my_wait(int x )
{
    unsigned int i , j ;
    for (i=0 ; i< x ; i++)
        for(j=0 ; j<255 ; j++);
}
```

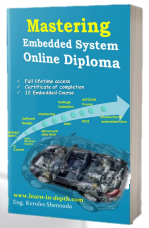
```
int main(void)
{
    unsigned ch ;

    //Enable CLock
    RCC_GPIOA_CLK_EN();
    RCC_GPIOB_CLK_EN();
    RCC_AFIO_CLK_EN();

    UART_Config uartCFG ;
    uartCFG.BaudRate = UART_BaudRate_115200 ;
    uartCFG.HwFlowCtl = UART_HwFlowCtl_NONE ;
    uartCFG.IRQ_Enable = UART_IRQ_Enable_NONE ;
    uartCFG.P_IRQ_CallBack = NULL ;
    uartCFG.Parity =UART_Parity__NONE ;
    uartCFG.Payload_Length = UART_Payload_Length_8B;
    uartCFG.StopBits = UART_StopBits__1 ;
    uartCFG.USART_Mode = UART_MODE_TX_RX ;

    MCAL_UART_Init(USART1, &uartCFG);
    MCAL_UART_GPIO_Set_Pins(USART1);

    /* Loop forever */
    while(1)
    {
        MCAL_UART_ReceiveData(USART1, &ch, enable);

        MCAL_UART_SendData(USART1, &ch, enable);

    }

}
```



CASE Study 1 with Debugging : using Polling mechanism



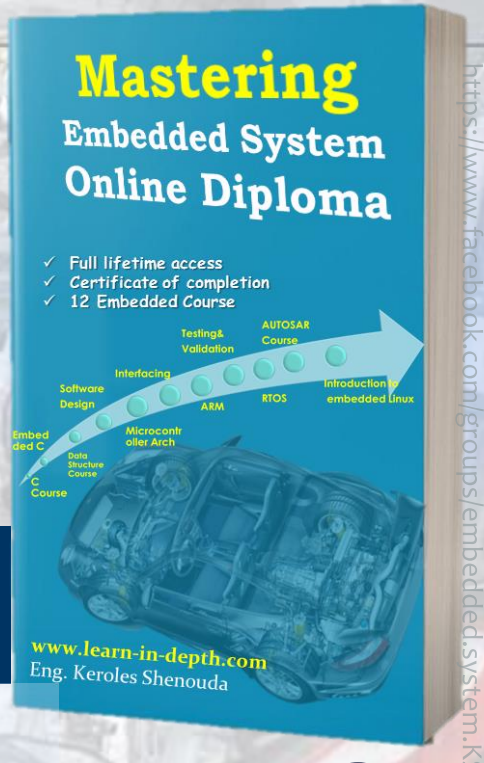
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

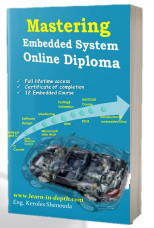
CASE Study 2 with Debugging : using Interrupt mechanism



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

CASE Study 2 with Debugging : using Interrupt mechanism



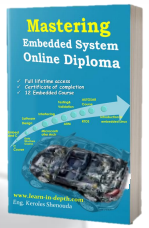
#LEARN_IN DEPTH

#Be_professional_in_embedded_system


```c
//Learn-in-depth
//Keroles Shenouda
//Mastering_Embedded System online diploma
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

#include "STM32F103x8.h"
#include "stm32f103x8_gpio_driver.h"
#include "lcd.h"
#include "keypad.h"
#include "stm32f103x8_EXTI_driver.h"
#include "stm32f103x8_USART_driver.h"


unsigned int IRQ_Flag = 0 ;
unsigned ch ;


void UART_IRQ_Callback (struct  UART_IRQ_Event* flag)
{

    MCAL_UART_ReceiveData(USART1, &ch, disable);
    MCAL_UART_SendData(USART1, &ch, enable);

}
```

```c
int main(void)
{

    //Enable CLock
    RCC_GPIOA_CLK_EN();
    RCC_GPIOB_CLK_EN();
    RCC_AFIO_CLK_EN();

    UART_Config uartCFG ;
    uartCFG.BaudRate = UART_BaudRate_115200 ;
    uartCFG.HwFlowCtl = UART_HwFlowCtl_NONE ;
    uartCFG.IRQ_Enable = UART_IRQ_Enable_RXNEIE ;
    uartCFG.P_IRQ_CallBack = UART_IRQ_Callback ;
    uartCFG.Parity =UART_Parity__NONE ;
    uartCFG.Payload_Length = UART_Payload_Length_8B;
    uartCFG.StopBits = UART_StopBits__1 ;
    uartCFG.USART_Mode = UART_MODE_TX_RX ;

    MCAL_UART_Init(USART1, &uartCFG);
    MCAL_UART_GPIO_Set_Pins(USART1);

    /* Loop forever */
    while(1)
    {
    }

}
```

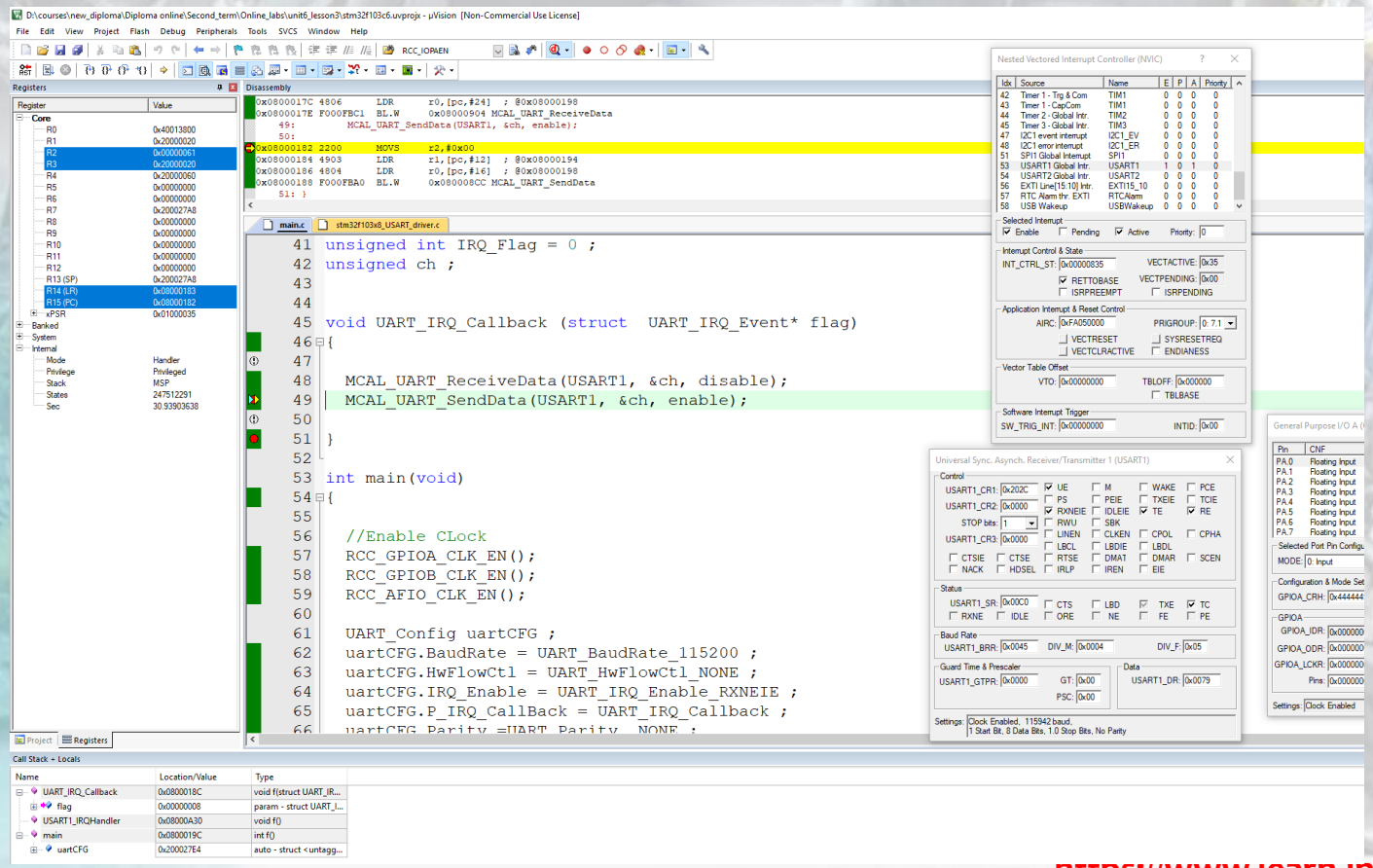
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



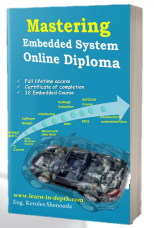
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

UART IRQ Mechanism



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



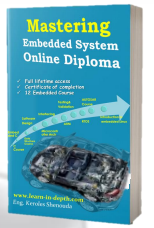
#LEARN_IN DEPTH

#Be_professional_in embedded_system

References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- Microprocessor: ATmega32 (SEE3223-10) http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller
- http://cs4hs.cs.pub.ro/wiki/roboticsisfun/chapter2/ch2_7_programming_a_microcontroller
- Embedded Systems with ARM **Cortex-M Microcontrollers in** Assembly Language and C Dr. Yifeng Zhu Third edition June 2018

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



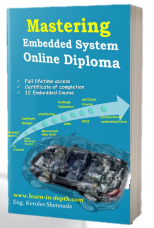
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

References

- http://techdifferences.com/difference-between-interrupt-and-polling-in-os.html
- http://www.bogotobogo.com/Embedded/hardware_interrupt_software_interrupt_latency_irq_vs_fiq.php
- Preventing Interrupt Overload Presented by Jiyong Park Seoul National University, Korea 2005. 2. 22. John Regehr, Usit Duogsaa, School of Computing, University.
- First Steps Embedded Systems Byte Craft Limited reference
- COMPUTER ORGANIZATION AND ARCHITECTURE DESIGNING FOR PERFORMANCE EIGHTH EDITION **William Stallings**
- **Getting Started with the Tiva™ TM4C123G LaunchPad Workshop**

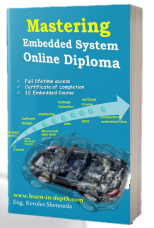
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- Tiva™ TM4C123GH6PM Microcontroller DATA SHEET
- Interrupts and Exceptions COMS W6998 Spring 2010
- THE **AVR MICROCONTROLLER**. AND EMBEDDED SYSTEMS Using Assembly and C. Muhammad Ali **Mazidi**.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- Microprocessor: ATmega32 (SEE3223-10) http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller
- AVR Microcontroller and Embedded Systems: Using Assembly and C (Pearson Custom Electronics Technology) 1st Edition https://www.amazon.com/AVR-Microcontroller-Embedded-Systems-Electronics/dp/0138003319

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/