AN6363

Application note

Introduction to clock requirements and calibration for STM32 MCUs

Introduction

This document describes the clock scheme and its capabilities for STM32 MCUs. Two oscillators, HSI and PSI, provide the flexibility and versatility required to support a wide range of applications and use cases. This document reviews the specific requirements of certain protocols included in the product peripherals and provides guidance to help users adjust the HSI if necessary. The STM32C5 is used as an example to explain peripheral clock requirements.

1 General information

This document applies to STM32 Arm® Cortex® based microcontrollers.

arm

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

2 Clock requirements per peripheral

This section outlines the protocol specification requirements that users must follow, as well as the flexibility available for configuration and settings.

For further details or specific register configurations, refer to the device reference manual.

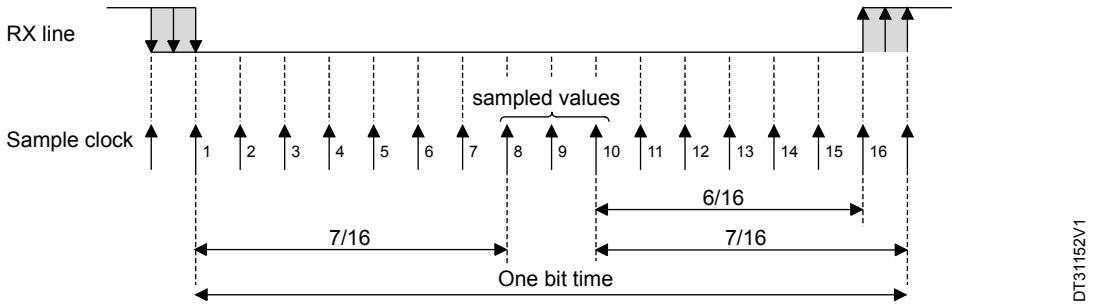
2.1 UART

2.1.1 Behavior

The universal asynchronous receiver transmitter (UART) is highly sensitive to clock inaccuracies due to its asynchronous mechanism.

The figure below shows that the UART sample clock must match the communication baud rate by a factor of 16 (or 8 if oversampling by 8). Consequently, a shift in the kernel clock frequency shifts the sampled value position, which can critically affect the last bits of the frame.

Figure 1. Data sampling when oversampling by 16



Two mechanisms within the UART peripheral mitigate the consequences of kernel clock inaccuracies:

- Autobaud rate: Allows the UART to build its sample clock frequency based on the first known transferred byte from the target, matching the transfer frequency of both devices.
- Noise detection: Uses a majority vote of three samples to detect errors. If one sample differs, an error is raised, allowing the user to trigger safety measures.

2.1.2 Requirements

The UART clock tolerance is calculated using the formula:

$DTRA + DQUANT + DREC + DTCL + DWU < USART \quad receiver \quad tolerance$

Where:

- DTRA: Deviation due to the transmitter error (including transmitter local oscillator deviation)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver local oscillator
- DTCL: Deviation caused by the transmission line, typically due to transceivers introducing asymmetry between low-to-high and high-to-low transition timings
- DWU: Is the error due to sampling point deviation when waking up from low-power mode

Receiver tolerance, determined by design, is also impacted by configuration settings such as character length (9, 10, or 11), oversampling configuration (8 or 16), baud rate, and the number of samples used for bit detection (ONEBIT setting). It typically ranges from approximately 2% to 4% and is affected by the baud rate (the higher the baud rate, the greater the impact of voltage level changes on the line) and noise on the supply and I/O ring.

2.1.3 Optimization

To ensure the most robust communication, consider:

- Aligning exact baud rates between elements to minimize quantization error.
- Limiting transmission line deviation by selecting appropriate transceivers and lines.

- Periodically relaunching autobaudrate calibration, especially with temperature changes.

If calibration using references like LSE, HSE, or USB SOF is not possible, consider alternative methods to ensure internal clock stability (refer to Section 4: HSI trimming strategy).

2.2 FDCAN

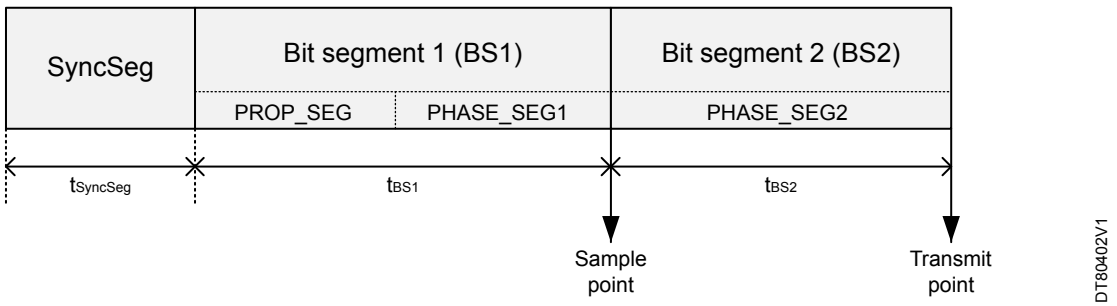
2.2.1 Bit timing and clock accuracy

The bit timing logic monitors the serial bus-line and performs sampling and adjustment of the sample point by synchronizing on the start-bit edge and resynchronizing on subsequent edges.

The bit time is divided into three segments:

- Synchronization segment (SYNC_SEG): a bit change is expected to occur within this time segment, having a fixed length of one-time quantum (1 × tq).
- Bit segment 1 (BS1): Defines the sample point location, programmable from 1 to 16 time quanta, and can be lengthened to compensate for positive phase drifts. It includes the PROP_SEG and PHASE_SEG1 of the CAN standard.
- Bit segment 2 (BS2): defines the transmit point location, programmable between one and height time quanta, and can be shortened to compensate for negative phase drifts. It represents the PHASE_SEG2 of the CAN standard.

Figure 2. Bit timing



The baud rate is the inverse of the bit time (baud rate = 1 / bit time), which is the sum of the three segments (see Figure 2):

$$bit \quad time = tSynSeg + tBS1 + tBS2$$

Where:

- For the nominal bit time:

$$tq = (NBRP[8:0]+1) \times tfdcan\_tq\_clk$$

$$tSyncSeg = 1 \times tq$$

$$tBS1 = tq \times (NTSEG1[7:0] + 1)$$

$$tBS2 = tq \times (NTSEG2[6:0] + 1)$$

Where NBRP[8:0], NTSEG1[7:0], and NTSEG2[6:0] bitfields belong to the FDCAN_NBTP register.

- For the data bit time:

$$tq = (DBRP[4:0] + 1) \times tfdcan\_tq\_clk$$

$$tSyncSeg = 1 \times tq$$

$$tBS1 = tq \times (DTSEG1[4:0] + 1)$$

$$tBS2 = tq \times (DTSEG2[3:0] + 1)$$

The (re)synchronization jump width (SJW) defines the maximum lengthening or shortening of bit segments and is programmable between one- and four-time quanta.

A valid edge is defined as the first transition in a bit time from dominant to recessive bus level, provided the master itself does not send a recessive bit.

If a valid edge is detected in BS1 instead of SYNC_SEG, BS1 is extended by up to SJW, so that the sample point is delayed.

Conversely, if a valid edge is detected in BS2 instead of SYNC_SEG, BS2 is shortened by up to SJW, so that the transmit point is moved earlier.

2.2.2 Oscillator tolerance

Oscillator tolerance in CAN or FDCAN systems is defined by five conditions related to nominal (conditions 1 and 2) and data speeds (conditions 3, 4, and 5). Data speed is the transition between the arbitration and data phases.

Conditions 1 and 2 apply to CAN operation. All conditions must be met for FDCAN operation.

The required tolerance is the minimum result from these conditions.

For more detailed information about oscillator tolerance and robustness of a CAN FD network, refer to this publication:

Robustness of a CAN FD Bus System – About Oscillator Tolerance and Edge Deviations, Dr. Arthur Mutter, Robert Bosch GmbH, 2013

Condition properties

Subscripts

- N: denotes variables of arbitration phase (Nominal speed)
- D: denotes variables of Data Phase (Data speed)

Formula properties

- *BRP_D,BRP_N: Baud Rate Prescaler*
- *SJW_D,SJW_N: Synchronization Jump Width*
- *PS2_D,PS2_N: Phase Segment 2*
- *PS1_D,PS1_N: Phase Segment 1*
- *PROP_SEG(N),PROP_SEG(D): Propagation segment*
- *BT_D,BT_N: Bit time duration*

In the equations, all properties are expressed in Time Quanta (TQ). Below is the method to calculate the various properties from the register values contained in the FDCAN_DBTP and FDCAN_NBTP registers.

How to calculate the properties:

$$BRP_N = NBRP + 1$$

$$SJW_N = NSJW + 1$$

$$PS1_N = NTSEG1 - PROP\_\mathrm{SEG(N)} + 1$$

$$PS2_N = NTSEG2 + 1$$

$$BT_N = NTSEG1 + NTSEG2 + 3$$

$$BRP_D = DBRP + 1$$

$$SJW_D = DSJW + 1$$

$$PS1_D = DTSEG1 - PROP\_SEG(D) + 1$$

$$PS2_D = DTSEG2 + 1$$

$$BT_D = DTSEG1 + DTSEG2 + 3$$

Condition equations

Condition 1: arbitration phase resynchronization

$$df = \frac{SJW_N}{2 \times 10 \times BT_N}$$

Condition 2: arbitration phase sampling bit succeeding own error flag

$$df = \frac{min(PS1_N, PS2_N)}{2 \times [13 \times BT_N - PS2_N]}$$

Condition 3: data phase resynchronization

$$df = \frac{SJW_D}{2 \times 10 \times BT_D}$$

Condition 4: arbitration phase sampling bit succeeding own error flag

$$df = \frac{min(PS1_N, PS2_N)}{2 \times \left[(6 \times BT_D - PS2_D) x \frac{BRP_D}{BRP_N} + 7 \times BT_N\right]}$$

Condition 5: switching from arbitration phase to data phase

$$df = \frac{SJW_D - max\left(0, \frac{BRP_N}{BRP_D} - 1\right)}{2 \times \left[(2 \times BT_N - PS2_N) \times \frac{BRP_N}{BRP_D} + PS2_D + 4 \times BT_D\right]}$$

2.2.3 Example for a CAN network

This first example calculates the required clock tolerance for a CAN network operating at 100 kbps nominal bit rate and 8 MHz clock. Refer to Table 1 for the network configuration.

Table 1. CAN network example configuration

Speed	Register configuration	Condition properties
Nominal (FDCAN_NBTP)	NBRP = 7 NSJW = 3 NTSEG1 = 4 NTSEG2 = 3 PROP_SEG(N) = 1	$BRP_N = 8$ $SJW_N = 4$ $PS1_N = 4$ $PS2_N = 4$ $BT_N = 10$

Result:

- Condition 1: *df = 2.0%*
- Condition 2: *df = 1.58%*

The clock tolerance requested is 1.58%. This is the largest possible oscillator tolerance in CAN operation.

2.2.4 Example for a FDCAN network

This second example calculates required clock tolerance for a FDCAN network operating at 250 kbps nominal bit rate and 1 Mbps data bit rate, clocked at 24 MHz. Refer to Table 2 for the network configuration.

- FDCAN clock frequency: 24 MHz
- Nominal bit rate: 250 kbps
- Data bit rate: 1000 kbps

Table 2. FDCAN network example configuration

Speed	Register configuration	Condition properties
Nominal (FDCAN_NBTP)	NBRP = 3 NSJW = 3 NTSEG1 = 18 NTSEG2 = 3 PROP_SEG(N) = 1	$BRP_N = 4$ $SJW_N = 4$ $PS1_N = 18$ $PS2_N = 4$ $BT_N = 24$
Data (FDCAN_DBTP)	DBRP = 1 DSJW = 3 DTSEG1 = 6 DTSEG2 = 3 PROP_SEG(D) = 1	$BRP_D = 2$ $SJW_D = 4$ $PS1_D = 6$ $PS2_D = 4$ $BT_D = 12$

Result:

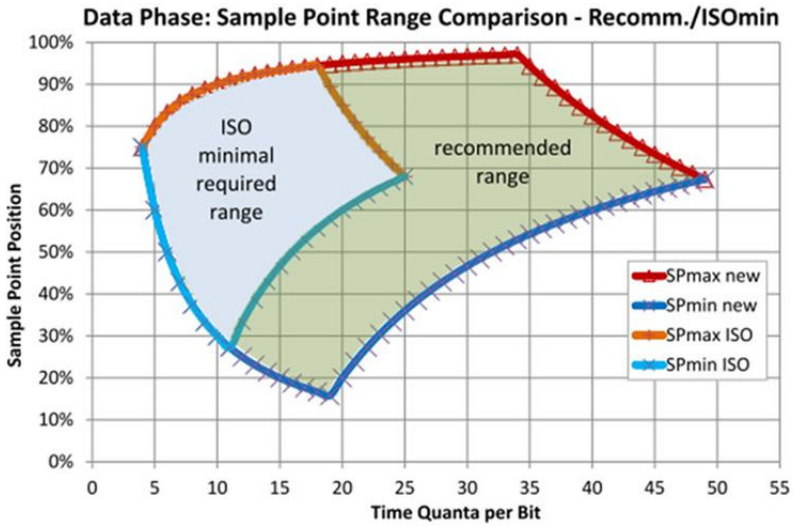
- Condition 1: *df = 0.83%*
- Condition 2: *df = 0.65%*
- Condition 3: *df = 1.68%*
- Condition 4: *df = 0.99%*
- Condition 5: *df = 1.07%*

The clock accuracy required by this network is 0.65%.

2.2.5 Conclusion

Clock accuracy required by CAN/FDCAN networks largely depends on speed configuration and sampling timing. STM32 internal clock accuracy is approximately ±1% at ambient temperature and ±3% over the full temperature range. Therefore, it is recommended to use an accurate external clock (directly or via PSI), since CAN/FDCAN clock accuracy requirements can be more stringent than internal clock accuracy.

Figure 3. Data phase: sample point range comparison



2.3 USB full-speed

As stated explicitly in the specification: “The full-speed data rate is nominally 12.000 Mb/s. For full-speed only functions, the required data-rate when transmitting (TFDRATE) is 12.000 Mb/s +/- 0.25% (2,500 ppm).”

This data rate is typically built from a 48 MHz kernel clock. Two scenarios must apply: host and device.

2.3.1 Host

The MCU leads the transfer rate; clock requirements remain unchanged. However, because the host initiates transfers, it cannot initially depend on the device's accuracy. An external reference is necessary to lock the oscillator in PLL mode (PSI + HSE for example) or to trim HSI accurately (HSI + CRS + LSE for example).

2.3.2 Device

The MCU can rely on the host start of frame (1 ms periodic USB signal) to trim the HSI144, eliminating the need for an external crystal and improving HSI accuracy.

2.4 Ethernet

Ethernet communication involves a media access control (MAC) layer interfacing with a Physical (PHY) layer. Clock requirements are essential for data integrity and synchronization. These requirements include support for 10BASE-T1S, an Ethernet standard used in automotive and industrial applications.

2.4.1 Clock requirements

Reference clock

The PHY requires a 25 MHz or 50 MHz reference clock for generating timing signals essential to data transmission and reception. For 10BASE-T1S, a 25 MHz reference clock is standard.

Fast Ethernet (100BASE-TX) and gigabit Ethernet (1000BASE-T) often use a 25 MHz clock, internally multiplied by the PHY to 125 MHz. Some systems may also use a 100 MHz reference clock.

MAC clock

The MAC layer operates using a system clock derived from the microcontroller’s main clock. For 100 Mbps and 1 Gbps Ethernet, the MAC typically requires a stable 100 MHz clock

Synchronization

The MAC and PHY must be synchronized to maintain the correct timing for data packets. This synchronization is achieved through the use of the reference clock provided to the PHY.

Clock tolerance and jitter

Ethernet standards typically allow a clock tolerance of ±100 ppm (parts per million) for the reference clock. This ensures that the clock remains within acceptable limits for reliable communication. Additionally, jitter should be minimized to prevent timing inaccuracies, which can affect data integrity, especially in time-sensitive applications like 10BASE-T1S.

Precision time protocol (PTP) and pulse per second (PPS) clocks

For applications requiring precise timing, such as industrial automation or telecommunications, PTP can be used to synchronize clocks throughout a network. PPS signals can also be employed for high-precision timekeeping, providing a reference that ensures synchronization accuracy.

2.4.2 Clock source

An external crystal oscillator is recommended for the reference clock to achieve the required accuracy and stability. This is crucial for minimizing jitter and ensuring consistent performance.

Clock configuration

The clock configuration must be set correctly in the microcontroller’s registers to match the PHY requirements. This includes setting the correct frequency and ensuring proper synchronization between the MAC and PHY.

Additional considerations for 10BASE-T1S

10BASE-T1S uses a multidrop physical layer, allowing multiple nodes to share the same communication medium. This requires precise clock synchronization to avoid collisions and ensure reliable data transmission. The PHY must support the specific signaling and timing requirements of 10BASE-T1S, which may include specific register settings and configuration parameters.

Conclusion

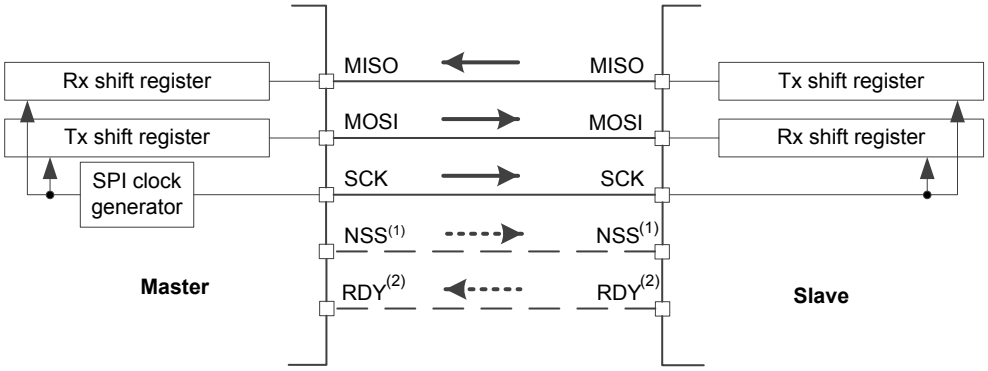
The clock requirements for the MAC to PHY link in Ethernet communication, including support for 10BASE-T1S, are critical for ensuring data integrity and synchronization. Using an accurate and stable external clock source is recommended to meet the stringent tolerance requirements imposed by Ethernet standards. Incorporating PTP or PPS clocks can further enhance synchronization precision in time-sensitive applications.

By adhering to these clock requirements and considering the specific needs of 10BASE-T1S, reliable Ethernet communication is ensured for STM32 microcontrollers.

2.5 SPI

Serial peripheral interface (SPI) is a synchronous communication protocol used for short-distance communication, primarily in embedded systems. The clock requirements for SPI are crucial for ensuring data integrity and synchronization between the master and slave devices.

Figure 4. Full-duplex single master/single slave application



2.5.1 SPI clock (SCK)

The SPI clock (SCK) is generated by the master device and is used to synchronize the data transmission and reception between the master and slave devices.

The frequency of the SPI clock can vary widely, typically ranging from a few kHz to several MHz, depending on the system requirements and the capabilities of the devices involved.

2.5.2 Clock configuration

The SPI clock frequency is typically set by configuring the SPI control registers in the master device. This involves setting a prescaler value to divide the master clock to achieve the desired SPI clock frequency.

2.5.3 Clock tolerance and jitter

Since SPI is a synchronous protocol, the clock accuracy is not a major concern. The master device controls the clock, ensuring that both master and slave are synchronized during data transmission.

However, minimizing jitter is still important to ensure stable and reliable communication, especially at higher frequencies. The clock signal should be clean and stable to avoid errors in data transmission.

2.5.4 Data transfer rate

The SPI data transfer rate is directly related to the SPI clock frequency. Higher clock frequencies allow faster data transfer rates, but may require careful consideration of signal integrity and timing constraints.

2.5.5 Synchronization

SPI is inherently synchronous, meaning that the clock signal is used to synchronize the data transmission and reception. This requires precise timing to ensure that data is correctly sampled and shifted.

In full duplex communication, special attention must be given at high transfer rate to the message from the slave over the received clock. Some time may be needed to align signals to ensure proper synchronization. To cover this constraint, the DRDS bit (Bit 24 in the SPI control register) can be set to introduce a data sampling delay on the master input (MISO). This delay compensates for the delay on the MISO line, such as delays caused by signal optical isolation.

- **DRDS = 0:** Sampling is performed on the configured SCK edge (no delay is applied).
- **DRDS = 1:** Sampling is postponed by a fixed delay equal to a half SCK cycle. This means sampling is performed just at the end of the bit validity interval on the MISO line.

Note: The DRDS setting does not impact other SCK management or the data frame communication format. When the delay is applied, it ensures more accurate sampling in the presence of line delays.

2.5.6 I2S and audio clock

I2S peripheral might require very accurate clocks with particular frequencies. A dedicated clock input is available to serve those particular requirements in place of AUDIOCLK input.

2.5.7 Conclusion

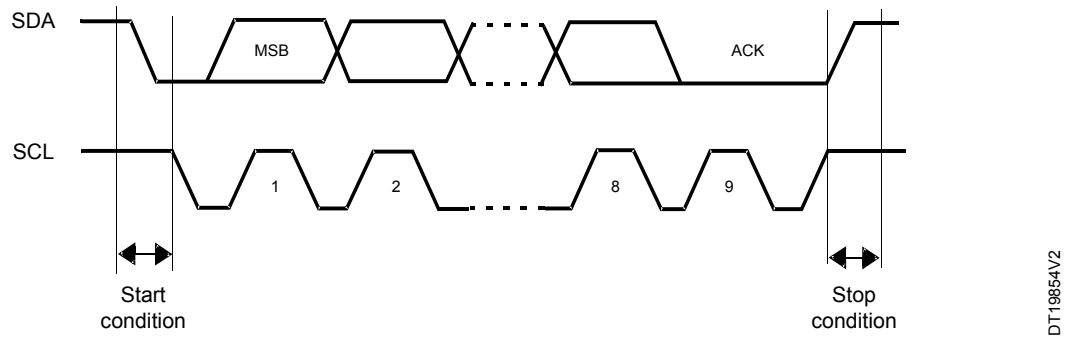
The clock requirements for SPI communication are essential for ensuring data integrity and synchronization between master and slave devices. Due to its synchronous nature, clock accuracy is not a major concern in SPI communication. Proper configuration of the SPI clock frequency, polarity, and phase is crucial for reliable communication. Minimizing jitter and ensuring stable clock signals can further enhance communication reliability, especially at higher frequencies.

By adhering to these clock requirements, reliable SPI communication is ensured for STM32 microcontrollers.

2.6 I2C

Inter-integrated circuit unit (I2C) is a synchronous, multi-controller/multi-target, packet-switched, single-ended, serial communication bus. It is widely used for connecting low-speed peripherals to microcontrollers. The clock requirements for I2C are crucial for ensuring data integrity and synchronization between the controller and target devices.

Figure 5. I²C-bus protocol



2.6.1 I²C-bus clock (SCL)

- The I²C-bus clock (SCL) is generated by the controller device and is used to synchronize data transmission and reception between the controller and target devices.
- The clock frequency can vary depending on the mode of operation: standard mode (up to 100 kHz), fast mode (up to 400 kHz), and fast mode plus (up to 1 MHz).

2.6.2 Clock configuration

- The I²C clock frequency is set by configuring the I2C unit control registers in the controller device. This involves setting appropriate timing parameters to achieve the desired clock frequency.
- The clock configuration must ensure that the setup and hold times for data and clock signals meet the I2C unit specification requirements.

2.6.3 Clock stretching

- I2C unit supports clock stretching, where the target device can hold the SCL line low to delay the controller, allowing more time for processing data. This feature provides flexibility and reduces constraints on clock accuracy, as it allows the target to manage its processing time effectively.
- Clock stretching ensures that the communication remains synchronized even if the target needs additional time, making I2C unit robust against timing variations.

2.6.4 Clock tolerance and jitter

- Due to its synchronous nature and the presence of clock stretching, I²C-bus is less sensitive to clock accuracy issues compared to asynchronous protocols. The controller controls the clock, ensuring synchronization.
- Minimizing jitter is still beneficial, especially in fast mode and fast mode plus, to maintain stable and reliable communication.

2.6.5 Synchronization

- I²C-bus is inherently synchronous, meaning that the clock signal is used to synchronize data transmission and reception. This synchronous nature, combined with clock stretching, reduces the need for precise clock accuracy, as the protocol can adapt to timing variations.

2.6.6 Fast mode and fast mode plus

- **Fast mode (FM):** Operates at speeds up to 400 kHz. It requires stricter timing constraints compared to Standard Mode, including shorter setup and hold times.
- **Fast mode plus (FM+):** Operates at speeds up to 1 MHz. This mode imposes even stricter timing requirements and may require additional considerations such as stronger pull-up resistors to ensure signal integrity.
- Both fast mode and fast mode plus benefit from the flexibility provided by clock stretching, allowing the system to handle higher speeds with reduced risk of timing errors.

2.6.7 Conclusion

The clock requirements for I2C unit communication are essential for ensuring data integrity and synchronization between controller and target devices. Due to its synchronous nature and the support for clock stretching, I²C-bus has reduced constraints on clock accuracy. Proper configuration of the I²C clock frequency and timing parameters is crucial for reliable communication. Minimizing jitter and ensuring stable clock signals can further enhance communication reliability, especially in fast mode and fast mode plus.

By adhering to these clock requirements and considering the specific needs of an application, reliable I2C unit communication is ensured for STM32 microcontrollers.

2.7 I3C

The improved inter-integrated circuit (I3C) is a synchronous communication protocol designed to enhance and extend the capabilities of the traditional I²C protocol. It offers higher data rates and more efficient communication, making it suitable for a wide range of applications.

The clock requirements for I3C are crucial for ensuring data integrity and synchronization between controller and target devices. This chapter discusses the impact of clock jitter and inaccuracy on I3C performance and the potential distortion it might introduce.

2.7.1 I3C clock (SCL)

- The I3C clock (SCL) is generated by the controller device and is used to synchronize data transmission and reception between the controller and target devices.
- The clock frequency supports up to 12.5 MHz in standard mode.

2.7.2 Clock configuration

- The I3C clock frequency is set by configuring the I3C control registers in the controller device. This involves setting appropriate timing parameters to achieve the desired clock frequency.
- The clock configuration must ensure that the setup and hold times for data and clock signals meet the I3C specification requirements.

2.7.3 Clock tolerance and jitter

- While I3C is a synchronous protocol, minimizing jitter is important to ensure stable and reliable communication, especially at higher frequencies. The clock signal should be clean and stable to avoid errors in data transmission.

- Clock jitter refers to the small, rapid variations in the clock signal's timing. In the context of I3C, jitter can impact the accuracy of data transmission and reception. When operating at high data rates, even small amounts of jitter can cause significant errors in the sampled values.

2.7.4 Clock inaccuracy

- Clock inaccuracy refers to deviations in the clock frequency from its nominal value. While slight inaccuracies may be tolerable in some applications, significant deviations can affect the timing of the communication process.

2.7.5 Synchronization

- I3C is inherently synchronous, meaning that the clock signal is used to synchronize data transmission and reception. This synchronous nature reduces the need for precise clock accuracy, as the protocol can adapt to timing variations.
- However, in high-speed modes, precise timing is still required to ensure that data is correctly sampled and shifted.

2.7.6 Sampling distortion

- Distortion in I3C communication can arise from both jitter and clock inaccuracy. The primary types of distortion include:
- **Timing errors:** Caused by variations in the clock signal, leading to incorrect sampling of data bits.
- **Bit errors:** Occur when the timing inaccuracies cause bits to be misinterpreted, leading to data corruption.
- **Signal integrity issues:** High-frequency operation may introduce noise and interference, affecting the quality of the clock signal and leading to jitter and timing errors.

2.7.7 Minimizing jitter and inaccuracy

To minimize the effects of jitter and clock inaccuracy, consider the following strategies:

- **Use a stable clock source:** Employ high-quality crystal oscillators or external clock sources with low jitter and high stability.
- **Optimize clock configuration:** Configure the I3C clock settings to balance the trade-offs between speed and accuracy. Ensure that the clock frequency is appropriate for the data rate required.
- **Shielding and filtering:** Implement proper shielding and filtering techniques to reduce noise and interference that can contribute to jitter.

2.7.8 Comparison with I^2C-bus

Clock frequency

- I^2C-bus operates at lower frequencies (Standard Mode up to 100 kHz, Fast Mode up to 400 kHz, and Fast Mode Plus up to 1 MHz).
- I3C supports higher frequencies, typically up to 12.5 MHz, enabling faster data rates.

Clock stretching

- I^2C-bus supports clock stretching, allowing the target to hold the clock line low to delay the controller and manage processing time.
- I3C does not rely on clock stretching, instead using in-band interrupts and other mechanisms to handle communication efficiently.

Jitter and inaccuracy

- Both protocols benefit from minimizing jitter and ensuring stable clock signals, but the higher speeds of I3C make it more sensitive to these factors.
- The synchronous nature of both protocols means that clock accuracy is less critical compared to asynchronous protocols, but maintaining low jitter is still important for reliable communication.

2.7.9 Conclusion

The clock requirements for I3C communication are essential for ensuring data integrity and synchronization between controller and target devices. Clock jitter and inaccuracy can introduce significant distortion in the transmitted and received data, affecting the overall performance of the I3C protocol. By minimizing jitter and ensuring a stable and accurate clock source, you can enhance the precision and quality of I3C communication.

By adhering to these clock requirements and considering the specific needs of an application, reliable I3C communication is ensured for STM32 microcontrollers.

2.8 ADC

Analog to Digital Converters (ADCs) are crucial components in embedded systems, used to convert analog signals into digital data for processing. The clock requirements for ADCs are essential for ensuring accurate and reliable sampling of analog signals. This chapter discusses the impact of clock jitter and inaccuracy on ADC performance and the potential distortion it might introduce.

2.8.1 ADC clock (ADCCLK)

- The ADC clock (ADCCLK) drives the sampling and conversion processes of the ADC. It determines the rate at which the analog signal is sampled and converted into digital data.
- The frequency of the ADC clock can vary depending on the resolution and speed requirements of the application. Higher clock frequencies enable faster sampling rates but may introduce more noise and require careful management of jitter.
- The ADC is sensitive to clock duty cycle, more especially the minimum pulse width. When the input frequency is much lower than the maximum supported by the ADC, the ADC tolerates duty cycle distortion. However, when the input frequency approaches the maximum supported frequency, the ADC becomes sensitive to duty cycle distortion. For more information, refer to the product datasheet.

2.8.2 Clock jitter

- Clock jitter refers to the small, rapid variations in the clock signal's timing. In the context of ADCs, jitter can significantly impact the accuracy of the sampled data.
- When sampling high-frequency signals, even small amounts of jitter can cause significant errors in the sampled values. This is because the exact moment of sampling may vary, leading to inaccuracies in capturing the true amplitude of the analog signal.
- The impact of jitter is more pronounced at higher frequencies and higher resolutions. For high-precision applications, minimizing clock jitter is crucial to ensure accurate sampling.

2.8.3 Clock inaccuracy

- Clock inaccuracy refers to deviations in the clock frequency from its nominal value. While slight inaccuracies may be tolerable in some applications, significant deviations can affect the timing of the sampling process.
- Inaccurate clock frequencies can lead to non-uniform sampling intervals, introducing distortion in the sampled signal. This distortion can manifest as aliasing, where high-frequency components are incorrectly represented in the sampled data.

2.8.4 Sampling distortion

Distortion in ADC sampling can arise from both jitter and clock inaccuracy. The primary types of distortion include:

- **Quantization noise:** The inherent error introduced when converting an analog signal to a discrete digital value. This noise is always present but can be exacerbated by jitter and clock inaccuracies.
- **Aliasing:** Occurs when the sampling rate is insufficient to capture the high-frequency components of the analog signal accurately. Jitter and clock inaccuracies can worsen aliasing effects.
- **Harmonic distortion:** Introduced when the sampled signal contains harmonics that are not accurately captured due to timing variations. This distortion can degrade the quality of the digital representation of the analog signal.

2.8.5 Minimizing jitter and inaccuracy

To minimize the effects of jitter and clock inaccuracy, consider the following strategies:

- **Use a stable clock source:** Employ high-quality crystal oscillators or external clock sources with low jitter and high stability.
- **Optimize clock configuration:** Configure the ADC clock settings to balance the trade-offs between speed and accuracy. Ensure that the clock frequency is appropriate for the resolution and sampling rate required.
- **Shielding and filtering:** Implement proper shielding and filtering techniques to reduce noise and interference that can contribute to jitter.

2.9 Timers

Timers are essential peripherals in microcontrollers, used for a wide range of applications such as generating precise time delays, measuring time intervals, and producing PWM signals. The clock requirements for timers are crucial for ensuring accurate and reliable operation. This chapter discusses the various use cases of timers in microcontrollers and the potential side effects of clock inaccuracy in these applications.

2.9.1 Timer clock (TIMCLK)

- The timer clock (TIMCLK) is derived from the microcontroller's main system clock or an external clock source. It drives the timer's counting mechanism and determines the resolution and accuracy of the timer.
- The frequency of the timer clock can vary depending on the application requirements. Higher clock frequencies enable finer resolution and more precise timing, but may introduce more noise and require careful management of jitter.

2.9.2 Clock configuration

- The timer clock frequency is set by configuring the timer control registers in the microcontroller. This involves setting appropriate prescaler values to divide the main clock frequency to achieve the desired timer clock frequency.
- The configuration must ensure that the timer clock frequency meets the requirements of the specific application, balancing the trade-offs between resolution and accuracy.

2.9.3 Use cases and potential side effects of clock inaccuracy

Time delays

- Timers are commonly used to generate precise time delays in embedded systems. Accurate time delays are essential for tasks such as debouncing switches, creating timeouts, and implementing communication protocols.
- **Side effect of clock inaccuracy:** Clock inaccuracy can lead to incorrect time delays, causing timing mismatches and potentially leading to malfunctioning of the system. For example, a communication protocol may fail if the timeouts are not accurately maintained.

Time interval measurement

- Timers can be used to measure the duration of events or the time interval between two events. This is useful in applications such as frequency measurement, pulse width measurement, and event timing.
- **Side effect of clock inaccuracy:** Inaccurate clock frequencies can lead to incorrect time interval measurements, affecting the precision of the measurement. For instance, in frequency measurement, an inaccurate timer clock can result in incorrect frequency calculations.

Pulse Width Modulation (PWM)

- Timers are used to generate PWM signals, which are widely used for controlling the speed of motors, dimming LEDs, and generating audio signals.
- **Side effect of clock inaccuracy:** Clock inaccuracy can cause variations in the PWM duty cycle and frequency, leading to unstable motor speeds, flickering LEDs, or distorted audio signals.

Event counting

- Timers can be configured to count external events or pulses, useful in applications such as tachometers, flow meters, and rotary encoders.
- **Side effect of clock inaccuracy:** An inaccurate timer clock can affect the counting accuracy, leading to incorrect event counts and potentially erroneous system behavior.

Real-Time Clock (RTC)

- Some microcontrollers use timers to implement a real-time clock (RTC), which keeps track of the current time and date.
- **Side effect of clock inaccuracy:** Clock inaccuracy can cause the RTC to drift over time, leading to incorrect timekeeping. This can be problematic in applications that require precise time tracking, such as data logging and scheduling.

Capture/compare

- Timers can be used in capture/compare mode to capture the value of the timer at specific events or to generate output signals at precise intervals.

- **Side effect of clock inaccuracy:** Inaccurate clock frequencies can lead to incorrect capture values or imprecise output signal generation, affecting the timing accuracy of the system.

2.9.4 Conclusion

The clock requirements for timers are essential for ensuring accurate and reliable operation in various applications. Clock inaccuracy can introduce significant side effects, affecting the precision and functionality of the system. Proper configuration of the timer clock frequency and timing parameters is crucial for reliable timer operation. Minimizing jitter and ensuring a stable and accurate clock source can further enhance the performance of timers in microcontrollers.

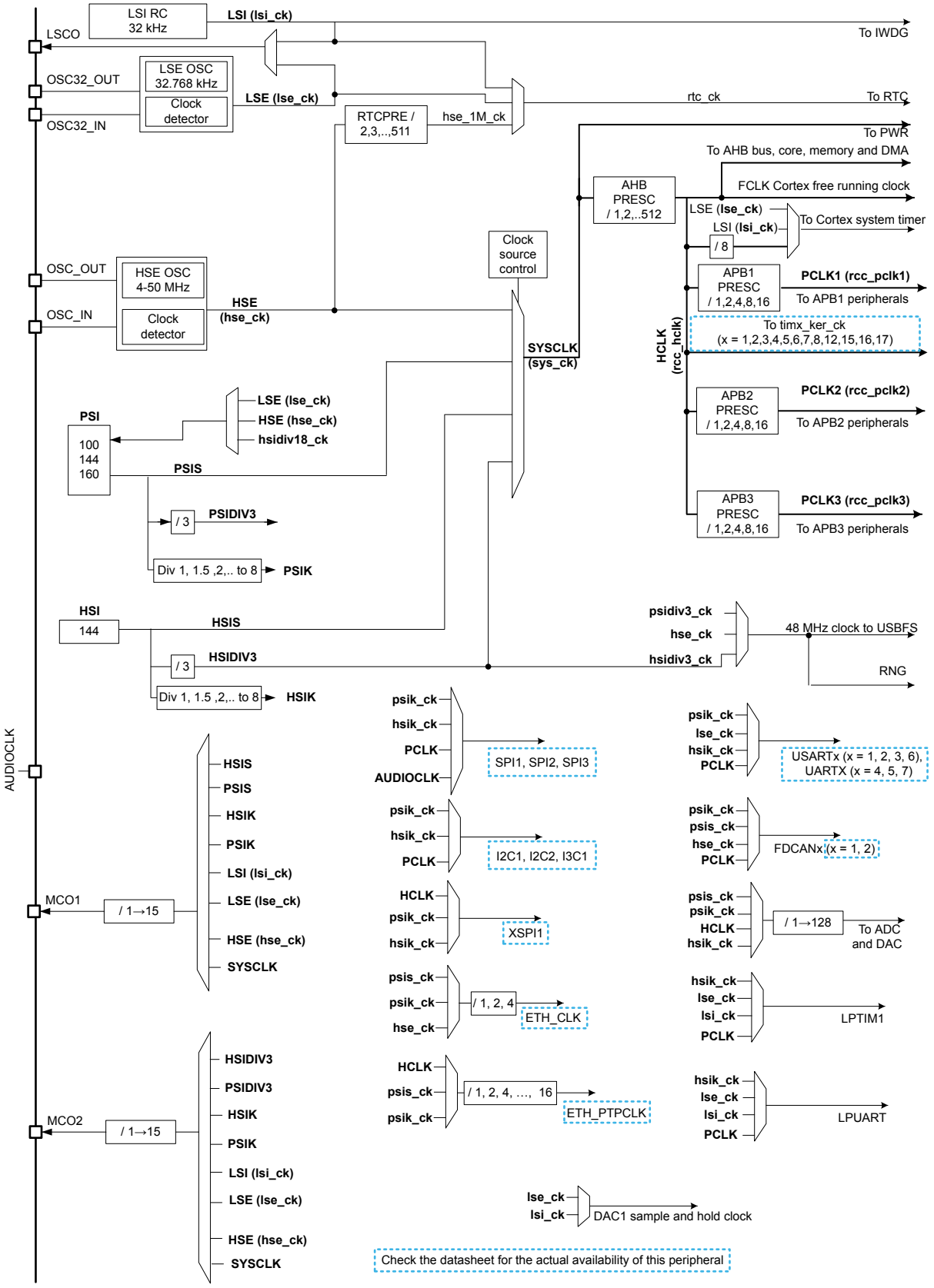
By adhering to these clock requirements and considering the specific needs of the application, reliable timer performance can be ensured with STM32 microcontrollers.

3 Example of the STM32C5 clock scheme

The STM32C5 series feature two internal oscillators.

- HSI: High speed internal oscillator. 144 MHz
- PSI: Precise speed internal oscillator. 100 / 144 / 160 MHz only operating when receiving a reference clock.

Figure 6. Clock tree



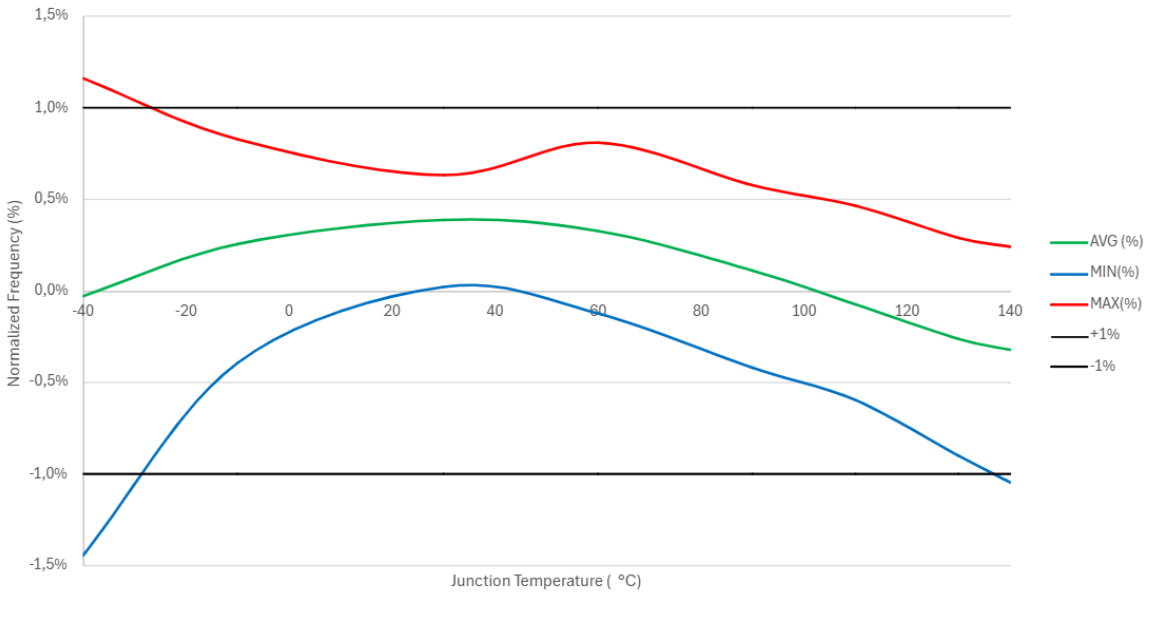
3.1 Accuracy achievable on HSI

The HSI is a user-trimmable internal oscillator, factory-trimmed to oscillate at a nominal frequency of 144 MHz. However, factors such as temperature, voltage variations, reflow mechanical stresses during production, or aging may affect this frequency.

Internal characterization ensures that the HSI frequency remains within a guaranteed accuracy range over a given temperature range. For exact accuracy data, refer to the datasheet of the selected product.

The frequency evolution over temperature follows a general trend, with minimum values occurring at the higher and lower ends of the supported temperature range. The maximum frequency is closer to the 30°C trim value. Refer to Figure 7 for details.

Figure 7. Typical HSI frequency evolution over temperature



This justifies that the default user trim value is not centered, to allow more compensation room for lower frequencies. The TRIM bit field reset value is 48, with a maximum range from 0 to 128.

3.2 Accuracy achievable on PSI

The PSI accuracy matches its reference signal. It operated as a phase-locked loop with a fixed output frequency.

3.3 Clock requirements in the case of STM32C5

The table below synthesize the clock requirements listed in Section 2: **Clock requirements per peripheral** and adapt them to the particular case of the STM32C5.

Table 3. Clock requirements per peripheral

Peripheral	Recommanded accuracy	Recommended clock source
UART	±1.5%	HSI (trimmed) or external crystal
FDCAN	≤0.65%	External crystal in direct or via PSI
USB FS host	±0.05%	48 MHz from external crystal (HSE or PSI)
USB FS device	±0.25%	48 MHz (HSE or HSI with CRS)
Ethernet	±100 ppm	External 25/50 MHz crystal
SPI	Low sensitivity	HSI or PSI
I2C	Low sensitivity	HSI or PSI
I3C	Jitter critical at 12.5 MHz	HSI or PSI
ADC	Jitter < 1 ps RMS for high speed, but depends on the application	External low-jitter source
Timers	Depends on the application	HSI or external

4 HSI trimming strategy

It is recommended to use PSI oscillator if an accurate clock source (HSE or LSE) is available in the application. However if the PSI is set to a different output frequency than 144 MHZ for specific use cases, maintaining HSI precision may be necessary. Additionally, if the HSE oscillator is not a multiple of 8MHz, it cannot be used with PSI.

The HSI trim mechanism is monotonic and linear. The trim steps are small enough to fulfill USB device requirements. For exact electrical characteristics, refer to the device datasheet.

4.1 HSI and CRS

The preferred trimming solution is the clock recovery system (CRS), a hardware-based, on-the-fly trim mechanism. It is designed to be used with USB device and trim the oscillator on accurate frequency. However, it can be extended to trim the HSI on external or internal reference clocks.

4.2 HSI and timers

Alternatively, many timers can receive a variety of reference clocks (through MCO or direct configuration), or external signals from the MCU. The HSI is used to clock the timer and set the frequency ratio against the reference. The HSI helps to trim the oscillator.

Some firmware overhead is needed to compute trim steps according to clock ratios.

5 Conclusion

The clock requirements for STM32 microcontrollers vary significantly across different protocols and applications. Ensuring accurate and stable clock sources is critical for maintaining data integrity, synchronization, and overall system performance. While internal oscillators like HSI and PSI provide flexibility, external clock sources are often recommended for strict accuracy requirements, such as in Ethernet or USB applications.

For protocols like UART and FDCAN, careful configuration of clock parameters and periodic recalibration can mitigate the effects of clock drift and jitter. Similarly, ADCs and timers benefit from stable clock sources to minimize sampling distortion and timing inaccuracies.

By adhering to the clock requirements outlined in this document and leveraging the STM32 reference manual for specific configurations, developers can optimize their designs for robust and reliable performance. For applications requiring high precision, external oscillators must be considered, or advanced trimming mechanisms such as CRS might occasionally provide a solution.

Revision history

Table 4. **Document revision history**

Date	Version	Changes
02-Feb-2026	1	Initial release.

Contents

List of tables

List of figures