ST25 fast transfer mode embedded library

Introduction

The ST25 fast transfer mode (ST25FTM) library is an embedded software library enabling fast data transfer between an NFC reader and a dynamic tag. The dynamic tag features a low latency memory that can be read and written by both an NFC reader and an MCU connected to the dynamic tag.

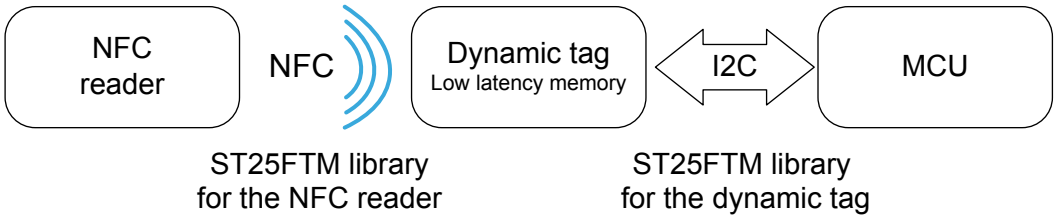
The ST25FTM library leverage on this low latency memory to exchange messages from the NFC reader to the MCU and the other way around.

The ST25FTM protocol has been designed to manage such data transfer, using as less meta-data as possible, while enabling error detection and recovery.

The ST25FTM library is an embedded implementation of the ST25FTM protocol that can be used on both an NFC reader and the MCU controlling the dynamic tag.

The ST25FTM library is part of the ST25 NFC embedded library package (STSW-ST25R-LIB). The STSW-ST25R-LIB embedded software provides middlewares and their associated examples that can be reused when developing an application with ST25R products.

Figure 1. Fast transfer mode overview



1 ST25FTM library examples

The ST25FTM library comes with examples running on the following hardware:

- X-NUCLEO-NFC08A1 featuring the ST25R3916B NFC reader.
- X-NUCLEO-NFC07A1 featuring the ST25DVxxKC dynamic tag.
- X-NUCLEO-NFC06A1 featuring the ST25R3916 NFC reader.
- X-NUCLEO-NFC05A1 featuring the ST25R3911 NFC reader.
- X-NUCLEO-NFC04A1 featuring the ST25DVxxK dynamic tag.
- X-NUCLEO-NFC03A1 featuring the ST25R95 NFC reader.

The NFC reader example starts polling for a ST25DV-I2C tag, and when detected, it initiates the transfer by transmitting a large amount of data. The dynamic tag example receives this data, switching the blue LED on, and once the transfer has completed, the dynamic tag example sends the data back to the NFC reader, switching the green LED on. In case of error the dynamic tag example switch the yellow LED on.

NFC reader and dynamic tag example sources are both providing an highlight on how to use the events (see Section 3.2: ST25FTM events management for more details). The event examples can be enabled in each demo by defining FTMDEMO_USE_EVENTS at compilation time. The use of events in one demo can be enabled/disabled independently from the other demo.

The NFC reader may be removed for a short period without jeopardizing the current transfer. If the NFC reader is kept away from the tag for a longer period, the transfer is reset and the NFC reader re-start polling for a ST25DV-I2C tag.

2 ST25FTM protocol

The low latency memory involved in the fast transfer mode has a defined length which is usually smaller than the length of the data to be transferred. The ST25FTM protocol splits the data to be transferred into messages fitting into the low latency memory.

The protocol manages the message chaining process using the first byte of all frames, each frame is referred to as a *packet*. This byte is named the control byte and each of its bit provides an information about the transfer state (see below description).

The ST25FTM protocol also provides the option to check the data integrity after a certain number of messages have been sent. The bunch of messages on which integrity is checked is called a *segment*.

ST25FTM control byte:

- b7 **message type**: set to 0 for a control byte.
- b6 **packet length**: when this bit is set, the packet length field is present in the message. This field indicates the length of the current message in bytes (including the CRC Bytes). The packet length field is necessary when the length of the message is smaller than the low-latency memory size. The Packet length field, when present, is appended after the control byte.
- b5 **segment end**: a CRC is present in this packet. The CRC is calculated on all the data present between “segment start” and “segment end” messages. An Acknowledge message is expected from the receiver as the next message.
- b4 **segment start**: The current message is the first message of a segment. The CRC computation must restart form this packet.
- b3-b2 **packet position**: These bits are used to indicate if the transfer requires several messages, and if yes it defines the position of the packet:
  - 00: The transfer length fit in a single message.
  - 01: First packet of a transfer. In that case, the Total Length field is present, next after the control byte. The Total Length field indicates the total payload length for the transfer in bytes.
  - 10: Middle packet of the transfer.
  - 11: End packet, finishing the transfer.
- b1 **segment id**: This bit toggles each time a segment has been successfully acknowledged, to facilitate detection of a retransmitted segment.
- b0: This bit indicates if the packet belongs to a segment.

When the transmitter has emitted a message with the segment end bit set (and the CRC was appended), the receiver is expected to reply with an acknowledge message. This acknowledge message is identified by its very first byte which is called the Status Byte.

ST25FTM Status byte

- b7 **message type**: set to 1 for a Status Byte.
- b6-b2: reserved for future use.
- b1-b0 **status**: acknowledge response for the received Segment.
  - 00 segment OK: the data integrity has been successfully checked (CRC).
  - 01 segment Error: the data integrity has failed.

Upon the reception of a successful acknowledge message, the transmitter continues by transmitting the first packet of the next Segment, toggling the segment Id bit from the control byte.

In case of unsuccessful acknowledge, the transmitter restarts and sends the previous segment again.

3 ST25FTM library

The ST25FTM library provides an implementation of the protocol described in Section 2: ST25FTM protocol that can be run on an MCU. The ST25FTM API can be called by an embedded application to transfer data using the fast transfer mode of a dynamic tag. The FTM transfer runs in a non-blocking way and progresses when the application runs the ST25FTM worker.

3.1 ST25FTM API

The ST25FTM library offers an API to the application to control the FTM transfers. This API is implemented in the st25ftm_protocol.c source file.

Table 1. ST25FTM_GetVersion function

Function	uint32_t ST25FTM_GetVersion(void)
Parameters	No parameters.
Returns	The current ST25FTM component version; formatted xxyyzz00 (with xx: major version, yy: minor version, zz: micro version)
Description	This function returns the current ST25FTM component version.

Table 2. ST25FTM_ST25FTM_Initialize function

Function	ST25FTM_Error_t ST25FTM_Initialize(void * initHandle)
Parameters	initHandle: a pointer to the structure potentially used by the ST25FTM_DeviceInit() interface function; can be NULL if not needed in ST25FTM_DeviceInit()
Returns	Error codes: • ST25FTM_OK: Init is Ok • ST25FTM_ERROR: Init failed
Description	This function initializes both: • the ST25FTM library itself. • the device, by calling the Interface function ST25FTM_DeviceInit() *Note: This function replaces the deprecated function ST25FTM_Init()*

Table 3. ST25FTM_Release function

Function	ST25FTM_Error_t ST25FTM_Release(void * initHandle)
Parameters	initHandle: a pointer to the structure potentially used by the ST25FTM_DeviceRelease() interface function; can be NULL if not needed in ST25FTM_DeviceRelease()
Returns	Error codes: • ST25FTM_OK: Release is Ok • ST25FTM_ERROR: Release failed
Description	This function un-initializes both: • The device, by calling the Interface function ST25FTM_DeviceRelease() • The CRC handler, by calling the Interface function ST25FTM_CRC_Release() • The ST25FTM library itself

Table 4. ST25FTM_SendStart function

Function	ST25FTM_Error_t ST25FTM_SendStart(uint8_t* data, uint32_t length, ST25FTM_Send_Ack_t ack)
Parameters	data: a pointer to the data buffer to be transmitted. length: number of bytes to be transmitted. ack: it enables hand-checks during the transmitter. • ST25FTM_SEND_WITHOUT_ACK: no acknowledge required. • ST25FTM_SEND_WITH_ACK: acknowledge is required. *Note: this function replaces the deprecated function ST25FTM_SendCommand(); related callback, if any, is now registered by calling ST25FTM_RegisterEvent(EVENT_FTM_TX_NEW_PKT, ...)*
Returns	Error codes: • ST25FTM_OK: Start is Ok • ST25FTM_ERROR: Start failed • ST25FTM_BUSY: Peer is busy • ST25FTM_TIMEOUT: connection timeout
Description	The function initializes a transfer to the peer device.

Table 5. ST25FTM_ReceiveStart function

Function	ST25FTM_Error_t ST25FTM_ReceiveStart(uint8_t* data, uint32_t maxLength, uint32_t *length)
Parameters	data: a pointer to the data buffer. It is used for data reception. maxLength: maximum number of bytes that can be received in the buffer length: a pointer to a word used to return the number of bytes actually read. *Note: this function replaces the deprecated function ST25FTM_ReceiveCommand(); related callback, if any, is now registered by calling ST25FTM_RegisterEvent(EVENT_FTM_RX_NEW_PKT, ...)*
Returns	Error codes: • ST25FTM_OK: Start is Ok • ST25FTM_ERROR: Start failed • ST25FTM_BUSY: Peer is busy • ST25FTM_TIMEOUT: connection timeout
Description	The function initializes a reception from the peer device.

Table 6. ST25FTM_Runner function

Function	uint8_t ST25FTM_Runner(void)
Parameters	No parameters.
Returns	Return the ST25FTM runner status: • 0x00: Runner has no more state machine transition to handle • 0x01: Runner has still a state machine transition to handle The returned value helps the application: • either to let the ST25FTM component internally looping on Runner until no more state machine transition is detected (RunnerPolicy set to ST25FTM_RUNNER_LOOP_INTERNALLY) or • to handle by itself the loops on Runner with regards to multitasks (RunnerPolicy set to ST25FTM_RUNNER_LOOP_WITHIN_APPLICATION). See the description of the API ST25FTM_SetRunnerPolicy() for more details.
Description	This function is the runner for the ST25FTM library. It must be run regularly to let the library send or receive the data.

Table 7. ST25FTM_SetRunnerPolicy function

Function	ST25FTM_Error_t ST25FTM_SetRunnerPolicy(uint8_t policy)
Parameters	policy: defines the Runner policy • ST25FTM_RUNNER_LOOP_INTERNALLY: the ST25FTM component Runner will internally loop until no more state machine transition is detected internally • ST25FTM_RUNNER_LOOP_WITHIN_APPLICATION: the ST25FTM component Runner will immediately return after current state processing
Returns	Error codes: • ST25FTM_OK: Parameter is Ok • ST25FTM_ERROR: Parameter is unknown
Description	This function defines the policy being used with the Runner. The policy must be adapted to the kind of application under execution: • ST25FTM_RUNNER_LOOP_INTERNALLY: the typical use of this option is with Events and ITs driven applications. • ST25FTM_RUNNER_LOOP_WITHIN_APPLICATION: the typical use of this option is with applications implementing an infinite loop on Runner.

Table 8. ST25FTM_RegisterEvent function

Function	ST25FTM_Error_t ST25FTM_RegisterEvent(ST25FTM_Protocol_Event_t event, ftm_events_cb_t event_cb, void *userData)
Parameters	event: registered event: • EVENT_FTM_BACK_TO_IDLE: state machine goes back to IDLE state • EVENT_FTM_RX_NEW_FRAME: very first packet is received • EVENT_FTM_RX_WAIT_ACK_READ: waiting the RX ACK to be read is under process • EVENT_FTM_RX_NEW_PKT: a new packet is received • EVENT_FTM_RX_NEW_SEGMENT: a new validated segment is received • EVENT_FTM_RX_DONE: full data is received and validated • EVENT_FTM_TX_WAIT_READ: waiting the TX data to be read is under process • EVENT_FTM_TX_WAIT_ACK_READ: waiting the TX ACK to be read is under process • EVENT_FTM_TX_NEW_PKT: a new packet is transmitted • EVENT_FTM_TX_NEW_SEGMENT: a new segment is transmitted • EVENT_FTM_TX_DONE: full data is transmitted • EVENT_FTM_ERROR: reception/transmission error raised event_cb: pointer on callback function (see below); NULL: disables the event. userData: pointer on event data handled by the user application (see below). The parameter event_cb is formatted as follows: typedef void (*ftm_events_cb_t)(ST25FTM_Protocol_Event_t event, ST25FTM_EventData_t *eventData, void *userData); The parameter userData will not be processed within the ST25FTM component, this parameter will be passed as is within the related callback leaving the user application handle it directly in the callback.
Returns	Error codes: • ST25FTM_OK: Parameter is Ok • ST25FTM_ERROR: Event is unknown
Description	This function enables a FTM event and registers it related callback.

Table 9. ST25FTM_Status function

Function	ST25FTM_State_t ST25FTM_Status(void)
Parameters	No parameters.
Returns	Return the current state of the ST25FTM lib (ST25FTM_State_t enum) ST25FTM_IDLE: No communication on-going. ST25FTM_WRITE: A transmission is on-going. ST25FTM_READ: A reception is on-going.
Description	This function returns the current state of the ST25FTM library.

Table 10. ST25FTM_SetTxFrameMaxLength function

Function	void ST25FTM_SetTxFrameMaxLength(uint32_t len)
Parameters	len: set the maximum message length (in bytes).
Returns	No return value.
Description	This function sets the maximum message length for the transmission.

Table 11. ST25FTM_SetRxFrameMaxLength function

Function	void ST25FTM_SetRxFrameMaxLength(uint32_t len)
Parameters	Len: set the maximum message length (in bytes).
Returns	No return value.
Description	This function sets the maximum message length for the reception.

Table 12. ST25FTM_GetTxFrameMaxLength function

Function	uint32_t ST25FTM_GetTxFrameMaxLength(void)
Parameters	No parameter.
Returns	Return the current maximum message length (in bytes) for the transmission.
Description	This function returns the maximum message length for the transmission.

Table 13. ST25FTM_GetRxFrameMaxLength function

Function	uint32_t ST25FTM_GetRxFrameMaxLength(void)
Parameters	No parameter.
Returns	Return the current maximum message length (in bytes) for the reception.
Description	This function returns the maximum message length for the reception.

Table 14. ST25FTM_SetTxTrim function

Function	void ST25FTM_SetTxTrim(uint32_t duration)
Parameters	duration: in ms of delay before the transmission starts.
Returns	No return value
Description	Sets a delay before the transmission duly starts leaving some time to receiver to set up appropriately

Table 15. ST25FTM_GetTxTrim function

Function	uint32_t ST25FTM_GetTxTrim(void)
Parameters	No parameter
Returns	returns the duration in ms before the transmission starts.
Description	Get the duration in ms before the transmission starts.

Table 16. ST25FTM_IsNewFrame function

Function	uint8_t ST25FTM_IsNewFrame(void)
Parameters	No parameter.
Returns	Return 1 the first time this function is called after a reception began, 0 if not.
Description	This function is used to let the application knows that a reception has started. *Note: This function may be replaced with the use of event: EVENT_FTM_RX_NEW_FRAME*

Table 17. ST25FTM_GetFieldState function

Function	ST25FTM_Field_State_t ST25FTM_GetFieldState(void)
Parameters	No parameter.
Returns	Return the current field state (ST25FTM_Field_State_t enum): ST25FTM_FIELD_OFF: the RF field is off. ST25FTM_FIELD_ON: the RF field is on.
Description	This function returns the current field state.

Table 18. ST25FTM_GetTransferProgress function

Function	uint32_t ST25FTM_GetTransferProgress(void)
Parameters	No parameter.
Returns	Return the current transfer progress in percentage from 0 to 100.
Description	This function computes and returns the progress of the current transfer.

Table 19. ST25FTM_GetAvailableDataLength function

Function	uint32_t ST25FTM_GetAvailableDataLength(void)
Parameters	No parameter.
Returns	Return the number of received bytes available (and validated).
Description	This function returns the number of received bytes.

Table 20. ST25FTM_GetReadBufferOffset function

Function	uint8_t ST25FTM_GetReadBufferOffset(uint8_t *dst, uint32_t length)
Parameters	dst: buffer to copy the data. length: number of bytes to be read.
Returns	Return the status: 0 if success, 1 if not.
Description	This function reads the specified number of bytes from the reception buffer.

Table 21. ST25FTM_ReadBuffer function

Function	uint8_t ST25FTM_ReadBuffer(length)
Parameters	length: number of bytes to be transmitted.
Returns	Return the offset of the next byte to read.
Description	This function gets the current offset in the command of the next byte, which is read with ST25FTM_ReadBuffer.

Table 22. ST25FTM_GetTotalLength function

Function	uint32_t ST25FTM_GetTotalLength(void)
Parameters	No parameter.
Returns	Return the total length of the current transfer (in bytes).
Description	This function returns the total length of the current transfer (in bytes), including the CRC.

Table 23. ST25FTM_GetRetryLength function

Function	uint32_t ST25FTM_GetRetryLength(void)
Parameters	No parameter.
Returns	Return the length of data that has been resent (in bytes).
Description	This function returns the length of data that has been resent (in bytes).

Table 24. ST25FTM_IsReceptionComplete function

Function	uint8_t ST25FTM_IsReceptionComplete(void)
Parameters	No parameter.
Returns	Return 1 if the reception has completed, 0 if not.
Description	This function returns the reception completion status. *Note: This function may be replaced with the use of event: EVENT_FTM_RX_DONE*

Table 25. ST25FTM_IsTransmissionComplete function

Function	uint8_t ST25FTM_IsTransmissionComplete(void)
Parameters	No parameter.
Returns	Return 1 if the transmission has completed, 0 if not.
Description	This function returns the transmission completion status. *Note: This function may be replaced with the use of event: EVENT_FTM_TX_DONE*

Table 26. ST25FTM_CheckError function

Function	uint8_t ST25FTM_CheckError(void)
Parameters	No parameter.
Returns	Return 1 if a fatal error occurred, 0 if not.
Description	This function returns the error status. *Note: This function may be replaced with the use of event: EVENT_FTM_ERROR*

Table 27. **ST25FTM_IsIdle function**

Function	uint8_t ST25FTM_IsIdle(void)
Parameters	No parameter.
Returns	Return 1 if the ST25FTM library is idle, 0 if a transfer is running.
Description	This function returns the ST25FTM library idle status. *Note: This function may be replaced with the use of event: EVENT_FTM_BACK_TO_IDLE*

Table 28. **ST25FTM_Reset function**

Function	void ST25FTM_Reset(void)
Parameters	No parameter.
Returns	No return value.
Description	This function resets the ST25FTM library state.

Table 29. **ST25FTM_SetTxSegmentMaxLength function**

Function	uint8_t ST25FTM_SetTxSegmentMaxLength(uint32_t length)
Parameters	Length: number of bytes to be transmitted.
Returns	The maximum value is ST25FTM_SEGMENT_LEN.
Description	This function sets the length of a segment in bytes.

Table 30. **ST25FTM_ResetTxSegmentMaxLength function**

Function	uint8_t ST25FTM_ResetTxSegmentMaxLength(uint32_t length)
Parameters	Length: number of bytes to be transmitted.
Returns	The maximum value is ST25FTM_SEGMENT_LEN.
Description	This function resets the length of a segment in bytes. The maximum value is ST25FTM_SEGMENT_LEN.

Table 31. **ST25FTM_GetTxSegmentMaxLength function**

Function	uint8_t ST25FTM_GetTxSegmentMaxLength(uint32_t length)
Parameters	Length: number of bytes to be transmitted.
Returns	The maximum value is ST25FTM_SEGMENT_LEN.
Description	This function gets the length of a segment in bytes.

3.2 ST25FTM events management

The ST25FTM components can raise some events at key steps of processing phases.

The user application can enable any of these events by registering a callback through the function ST25FTM_RegisterEvent() (see section ST25FTM_RegisterEvent() API description for more details).

3.2.1 Callback

The event is registered with a callback formatted as follows:

- void ftm_events_cb(ST25FTM_Protocol_Event_t event, ST25FTM_EventData_t *eventData, void *userData);

where:

- event: the current event for which the callback has been called.
- eventData: data related the current event (see Section 3.2.2: eventData parameter).
- userData: data pointer provided at registering phase.

The provided calback is executed in ST25FTM component context. The evenData parameter is updated by the ST25FTM component before calling the callback (see Section 3.2.2: eventData parameter for more details). The userData parameter is not modified in any case by the ST25FTM component, it is only passed through as is since the registration phase.

A special attention is required for EVENT_FTM_TX_NEW_PKT and EVENT_FTM_RX_NEW_PKT events with related callbacks. The callbacks related to these two events must at least operate a copy of sourcePtr data to targetPtr according to application constraints and buffers availability.

3.2.2 eventData parameter

Depending on event, the eventData parameter is structured as follows:

Table 32. EVENT_FTM_BACK_TO_IDLE event

Event	EVENT_FTM_BACK_TO_IDLE
Description	Raised when the state machine goes back to IDLE state
eventData structure fields	• bufferPtr = NULL • sourcePtr = NULL • bufferLength = 0

Table 33. EVENT_FTM_RX_NEW_FRAME event

Event	EVENT_FTM_RX_NEW_FRAME
Description	Raised when the very first packet is received
eventData structure fields	• bufferPtr = buffer start @ • sourcePtr = NULL • bufferLength = awaited data lg

Table 34. EVENT_FTM_RX_WAIT_ACK_READ event

Event	EVENT_FTM_RX_WAIT_ACK_READ
Description	Raised when waiting the RX ACK to be read is under process *Note: This event is sent twice:* • *when RX ACK wait starts (waitstatus=0x00)* • *when RX ACK is received (waitstatus=0x01)*
eventData structure fields	• bufferPtr = waitstatus @ • sourcePtr = NULL • bufferLength = 1 with waitstatus=0x00 if wait starts and waitstatus=0x01 if wait ends

Table 35. EVENT_FTM_RX_NEW_PKT event

Event	EVENT_FTM_RX_NEW_PKT
Description	Raised when a new packet is received
eventData structure fields	• bufferPtr = dest buffer @ • sourcePtr = src pkt buffer @ • bufferLength = data lg

Table 36. EVENT_FTM_RX_NEW_SEGMENT event

Event	EVENT_FTM_RX_NEW_SEGMENT
Description	Raised when a new validated segment is received
eventData structure fields	• bufferPtr = segment start @ • sourcePtr = NULL • bufferLength = segment data lg

Table 37. EVENT_FTM_RX_DONE event

Event	EVENT_FTM_RX_DONE
Description	Raised when full data is received and validated
eventData structure fields	• bufferPtr = buffer @ • sourcePtr = NULL • bufferLength = total data lg

Table 38. EVENT_FTM_TX_WAIT_READ event

Event	EVENT_FTM_TX_WAIT_READ
Description	Raised when waiting the TX data to be read is under process *Note: This event is sent twice:* • *when TX data to be read wait starts (waitstatus=0x00)* • *when TX data has been read (waitstatus=0x01)*
eventData structure fields	• bufferPtr = waitstatus @ • sourcePtr = NULL • bufferLength = 1 with waitstatus=0x00 if wait starts and waitstatus=0x01 if wait ends

Table 39. EVENT_FTM_TX_WAIT_ACK_READ event

Event	EVENT_FTM_TX_WAIT_ACK_READ
Description	Raised when waiting the TX ACK to be read is under process *Note: This event is sent twice:* • *when TX ACK to be read wait starts (waitstatus=0x00)* • *when TX ACK has been received (waitstatus=0x01)*
eventData structure fields	• bufferPtr = waitstatus @ • sourcePtr = NULL • bufferLength = 1 with waitstatus=0x00 if wait starts and waitstatus=0x01 if wait ends

Table 40. EVENT_FTM_TX_NEW_PKT event

Event	EVENT_FTM_TX_NEW_PKT
Description	Raised when a new packet is transmitted
eventData structure fields	• bufferPtr = dest buffer @ • sourcePtr = src pkt buffer @ • bufferLength = data lg

Table 41. **EVENT_FTM_TX_NEW_SEGMENT event**

Event	EVENT_FTM_TX_NEW_SEGMENT
Description	Raised when a new segment is transmitted
eventData structure fields	• bufferPtr = segment start @ • sourcePtr = NULL • bufferLength = segment data lg

Table 42. **EVENT_FTM_TX_DONE event**

Event	EVENT_FTM_TX_DONE
Description	Raised when full data is transmitted
eventData structure fields	• bufferPtr = buffer @ • sourcePtr = NULL • bufferLength = total data lg

Table 43. **EVENT_FTM_ERROR event**

Event	EVENT_FTM_ERROR
Description	Raised with any reception/transmission error where error: • ST25FTM_ERROR_RX_LENNULL: Received packet with size equals zero • ST25FTM_ERROR_RX_LENERROR: inconsistent length of received data • ST25FTM_ERROR_RX_LENLASTPKT: total received data length not aligned with expected length • ST25FTM_ERROR_RX_MAXREACHED: number of consecutive error with device has been reached while receiving data • ST25FTM_ERROR_TX_ACK: unknown ack value received • ST25FTM_ERROR_TX_MAXREACHED: number of consecutive error with device has been reached while transmitting data
eventData structure fields	• bufferPtr = last error @ • sourcePtr = NULL • bufferLength = 1

3.2.3 Events and ST25FTM APIs

The following ST25FTM API functions may be identically handled by events in application (and vice versa):

- ST25FTM_IsNewFrame(void) can be handled through the Event EVENT_FTM_RX_NEW_FRAME
- ST25FTM_IsReceptionComplete(void) can be handled through the Event EVENT_FTM_RX_DONE
- ST25FTM_IsTransmissionComplete(void) can be handled through the Event EVENT_FTM_TX_DONE
- ST25FTM_CheckError(void) can be handled through the Event EVENT_FTM_ERROR

3.3 ST25FTM middleware interface file

The ST25FTM library is delivered as a STM32Cube middleware. This means that the library is made independent from the hardware: MCU, NFC reader and dynamic tag. The header file Middlewares/ST/ST25FTM/Inc/st25ftm_interface.h defines the lowest layer API for the middleware. This defined API has to be implemented by the application for the targeted hardware. On top of this implemented APIs, the application has to provide the file `st25ftm_config_hal.h` which embeds all required macros (see below for macros definitions).

The interface files for the following STMicroelectronics NFC readers and dynamic tag are available in the ST25FTM library software package:

- The lowest layer API implementation file (Projects/STM32L476RG-Nucleo/Applications/X-NUCLEO-NFC0xA1/FTM/Src/st25r_st25dv-i2c_ftm.c) is the same for all NFC readers:
  - X-NUCLEO-NFC08A1 featuring the ST25R3916B NFC reader
  - X-NUCLEO-NFC06A1 featuring the ST25R3916 NFC reader
  - X-NUCLEO-NFC05A1 featuring the ST25R3911 NFC reader
  - X-NUCLEO-NFC03A1 featuring the ST25R95 NFC reader
- The interface header file (Projects/STM32L476RG-Nucleo/Applications/X-NUCLEO-NFC0xA1/FTM/Inc/st25ftm_config_hal.h) is the same for all NFC readers:
  - X-NUCLEO-NFC08A1 featuring the ST25R3916B NFC reader
  - X-NUCLEO-NFC06A1 featuring the ST25R3916 NFC reader
  - X-NUCLEO-NFC05A1 featuring the ST25R3911 NFC reader
  - X-NUCLEO-NFC03A1 featuring the ST25R95 NFC reader
- The lowest layer API implementation file (Projects/STM32L476RG-Nucleo/Applications/X-NUCLEO-NFC0xA1/FTM/NFC/Target/st25dv-i2c_ftm.c) is the same for all Dynamic Tags:
  - X-NUCLEO-NFC07A1 featuring the ST25DVxxKC dynamic tag
  - X-NUCLEO-NFC04A1 featuring the ST25DVxxK dynamic tag
- The interface header file (Projects/STM32L476RG-Nucleo/Applications/X-NUCLEO-NFC0xA1/FTM/Core/Inc/st25ftm_config_hal.h) is the same for all Dynamic Tags:
  - X-NUCLEO-NFC07A1 featuring the ST25DVxxKC dynamic tag
  - X-NUCLEO-NFC04A1 featuring the ST25DVxxK dynamic tag

The macros to be defined by the application in st25ftm_config_hal.h are as follows.

Table 44. ST25FTM_PACKED macro

Macro	Description
ST25FTM_PACKED(type)	Defines the packed attribute in correlation with used compiler

Table 45. ST25FTM_DEPRECATED macro

Macro	Description
ST25FTM_DEPRECATED	Defines the deprecated attribute in correlation with used compiler

Table 46. ST25FTM_MAX_NBR_OF_REGISTER_ERRORS macro

Macro	Description
ST25FTM_MAX_NBR_OF_REGISTER_ERRORS	Max number of consecutive DV/DVKC registers access errors

Table 47. ST25FTM_BUFFER_LENGTH macro

Macro	Description
ST25FTM_BUFFER_LENGTH	Length of the buffer used to store a single message data (aka packet)

Table 48. ST25FTM_SEGMENT_LEN macro

Macro	Description
ST25FTM_SEGMENT_LEN	Length of the buffer used to store unvalidated data

Table 49. ST25FTM_WAIT_TIMEOUT_IN_MS macro

Macro	Description
ST25FTM_WAIT_TIMEOUT_IN_MS	Timeout in ms before retransmiting a segment

Table 50. ST25FTM_TICK macro

Macro	Description
ST25FTM_TICK	Platform function to get the ms tick

Table 51. ST25FTM_DELAY macro

Macro	Description
ST25FTM_DELAY	Platform function to set a delay in ms

Table 52. ST25FTM_ENABLE_LOG macro

Macro	Description
ST25FTM_ENABLE_LOG	Defines if FTM logs are enabled (0: not enabled else logs are enabled)

Table 53. ST25FTM_LOG macro

Macro	Description
ST25FTM_LOG	Platform logger function

Table 54. ST25FTM_HEX2STR macro

Macro	Description
ST25FTM_HEX2STR	Platform function to stringify a data buffer

The interface for the ST25FTM middleware is as follow:

Table 55. ST25FTM_DeviceInit function

Function	ST25FTM_Error_t ST25FTM_DeviceInit(void * initHandle)
Parameters	initHandle: pointer to a structure used by the function.
Returns	ST25FTM_OK: success ST25FTM_ERROR: error ST25FTM_BUSY: component is busy ST25FTM_TIMEOUT: timeout
Description	This function initializes the device for the Fast Transfer Mode.

Table 56. ST25FTM_DeviceRelease function

Function	ST25FTM_Error_t ST25FTM_DeviceRelease(void * initHandle)
Parameters	initHandle: pointer to a structure used by the function.
Returns	ST25FTM_OK: success ST25FTM_ERROR: error ST25FTM_BUSY: component is busy ST25FTM_TIMEOUT: timeout
Description	This function releases the device for the Fast Transfer Mode.

Table 57. ST25FTM_CRC_Initialize function

Function	ST25FTM_Error_t ST25FTM_CRC_Initialize(void)
Parameters	No parameters.
Returns	ST25FTM_OK: success ST25FTM_ERROR: error ST25FTM_BUSY: component is busy ST25FTM_TIMEOUT: timeout
Description	This function initializes the CRC IP.

Table 58. ST25FTM_CRC_Release function

Function	ST25FTM_Error_t ST25FTM_CRC_Release(void)
Parameters	None
Returns	ST25FTM_OK: success ST25FTM_ERROR: error ST25FTM_BUSY: component is busy ST25FTM_TIMEOUT: timeout
Description	This function releases the CRC IP.

Table 59. ST25FTM_GetCrc function

Function	ST25FTM_Crc_t ST25FTM_GetCrc(uint8_t *data, uint32_t length, ST25FTM_crc_control_t control)
Parameters	data: pointer to a buffer of uint8_t on which to compute the CRC. length: the number of bytes in the data buffer. control: CRC phase control: • ST25FTM_CRC_START: CRC computation start • ST25FTM_CRC_END: CRC computation end • ST25FTM_CRC_ACCUMULATE: CRC computation continuing • ST25FTM_CRC_ONESHOT: CRC oneshot computation
Returns	The CRC value on 4 bytes.
Description	This function computes the CRC on the given data according to the CRC computation phase.

Table 60. ST25FTM_UpdateFieldStatus function

Function	void ST25FTM_UpdateFieldStatus(ST25FTM_Field_State_t * rfField)
Parameters	rfField: pointer on RF field to be updated by the function
Returns	No return value.
Description	This function updates the provided pointer with the current RF field value.

Table 61. ST25FTM_GetMessageOwner function

Function	ST25FTM_MessageOwner_t ST25FTM_GetMessageOwner(void)
Parameters	No parameters.
Returns	The owner of the message in the memory (ST25FTM_MessageOwner_t enum): ST25FTM_MESSAGE_EMPTY: the buffer is free. ST25FTM_MESSAGE_ME: busy with a message from this device. ST25FTM_MESSAGE_PEER: busy with a message from peer device. ST25FTM_MESSAGE_OWNER_ERROR: status unknown.
Description	This function detects the owner of the current message.

Table 62. ST25FTM_ReadMessage function

Function	ST25FTM_MessageStatus_t ST25FTM_ReadMessage(uint8_t *msg, uint32_t* msg_len)
Parameters	Msg: a buffer to be used to store the read message. Msg_len: a pointer to a 32 bits value, used to return the length of the message.
Returns	Status of the read operation (ST25FTM_Msg_Status_t enum): ST25FTM_MSG_OK: the message has been successfully read. ST25FTM_MSG_ERROR: an error occurred. ST25FTM_MSG_BUSY: the message has not been read, must retry.
Description	This function reads the peer device message from the FTM memory.

Table 63. ST25FTM_WriteMessage function

Function	ST25FTM_MessageStatus_t ST25FTM_WriteMessage(uint8_t* msg, uint32_t msg_len)
Parameters	msg: a buffer containing the message to be sent. msg_len: the length of the message.
Returns	Status of the read operation (ST25FTM_Msg_Status_t enum): ST25FTM_MSG_OK: the message has been successfully sent. ST25FTM_MSG_ERROR: an error occurred. ST25FTM_MSG_BUSY: the message has not been written, must retry.
Description	This function writes the message to the FTM memory.

3.4 ST25FTM Internal states

This section describes the different internal states of the ST25FTM states machines. The ST25FTM library has two state machines, one for the transmission and one for reception.

3.4.1 ST25FTM transmission state machine

The ST25FTM library transmission state machine works through the following states.

Table 64. ST25FTM transmission states

State	Description
ST25FTM_TX_IDLE	When the ST25FTM_SendStart function is called to provide data to be transmitted, the FTM transmission state machine initializes all the required state variables and the CRC IP.
ST25FTM_TX_INIT_TRANSMISSION	The ST25FTM transmission state machine prepares the next segment to be sent, according to the acknowledgment scheme selected (doing nothing/computing segment CRC).
ST25FTM_TX_WRITE_SEGMENT	The ST25FTM transmission state machine prepares the next message to be sent, according to its position in the segment.
ST25FTM_TX_WRITE_PKT	The ST25FTM transmission state machine actually writes the message to the FTM memory.
ST25FTM_TX_WAIT_READ	The ST25FTM transmission state machine waits until the message has been read by the peer device.
ST25FTM_TX_WAIT_ACK	The ST25FTM transmission state machine waits until the ACK is sent by the peer device.
ST25FTM_TX_READ_ACK	In case an ACK is expected from peer device, the ST25FTM transmission state machine poll to read the ACK message.
ST25FTM_TX_DONE	The transmission has successfully gone to its end.
ST25FTM_TX_ERROR	An unrecoverable error occurred.

3.4.2 ST25FTM reception state machine

The ST25FTM library reception state machine works through the following states.

Table 65. ST25FTM reception states

State	Description
ST25FTM_RX_IDLE	When the ST25FTM_ReceiveCommand function is called to provide data to be transmitted, the FTM reception state machine initializes all the required state variables and the CRC IP.
ST25FTM_RX_INIT_RECEPTION	The FTM reception state machine waits for an incoming message from the peer device.
ST25FTM_RX_READ_PKT	The FTM reception state machine reads the available message from the peer device. The message in unpacked and actions are taken according to the protocol metadata: initializing reception state variable, CRC checks, ...
ST25FTM_RX_WRITE_ACK	The FTM reception state machine writes an ACK message to the FTM memory.
ST25FTM_RX_WRITE_NACK	The FTM reception state machine writes a NACK message to the FTM memory.
ST25FTM_RX_WRITE_ERR	The FTM reception state machine writes an ERROR message to the FTM memory.
ST25FTM_RX_WAIT_ACK_READ	The FTM reception state machine waits for the ACK/NACK/ERROR message to be read by a peer device.
ST25FTM_RX_ACK_READ	The ACK/NACK/ERROR message has been read by a peer device.
ST25FTM_RX_DONE	The reception has successfully completed.
ST25FTM_RX_ERROR	An unrecoverable error occurred.

Revision history

Table 66. **Revision history**

Date	Revision	Changes
18-Jun-2020	1	Initial release
17-Dec-2021	2	Updated: • Section 2: ST25FTM protocol • Section 3.1: ST25FTM API • Section 3.4.1: ST25FTM transmission state machine Added: • Table 26. ST25FTM_CheckError function • Table 27. ST25FTM_IsIdle function • Table 28. ST25FTM_Reset function Deleted: • Table 15. ST25FTM_GetCryptoTime function
19-Mar-2024	3	Updated: • Section 1: ST25FTM library examples • Section 2: ST25FTM protocol • Section 3.1: ST25FTM API • Section 3.4.1: ST25FTM transmission state machine • Section 3.4.2: ST25FTM reception state machine Added: • Section 3.2: ST25FTM events management • Section 3.2.1: Callback • Section 3.2.2: eventData parameter • Section 3.2.3: Events and ST25FTM APIs

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved