# Slave:

## Transmitter:

### Slave\_TxLength < Master\_RxLength:

Currently, I have no idea what method to make the Master know that the Slave has no available data to transmit. If Master does not know that Slave has no more data available, it will waiting for new data from Slave forever.

### Slave\_TxLength > Master\_RxLength:

#### Polling Mode:

HAL\_I2C\_Slave\_Transmit return HAL\_ERROR because the Master send NACK while the Slave still has available data to send.

I don’t treat the response as HAL\_ERROR so I’ll change the return value to HAL\_OK.

1. /\* Check if a NACK is detected \*/
2. **if** (I2C\_IsAcknowledgeFailed(hi2c) != HAL\_OK)
3. {
4. **return** HAL\_ERROR;
5. }

=>

1. /\* Check if a NACK is detected \*/
2. **if** (I2C\_IsAcknowledgeFailed(hi2c) != HAL\_OK)
3. {
4. **return** HAL\_OK;
5. }

The condition stament above belong to I2C\_WaitOnTXEFlagUntilTimeout function locating in *stm32f4xx\_hal\_i2c.c* file.

~~The acctual amount of data which Slave transmitted:  
amount of data = Size - hi2c->XferCount~~

~~Size is the parameter of the HAL\_I2C\_Slave\_Transmit function.~~

#### Interrput Mode:

By default, Slave only perform HAL\_I2C\_SlaveTxCpltCallback right after it transmitted completely the Size of data which is inputed in HAL\_I2C\_Slave\_Transmit\_IT function.

Prior to HAL\_I2C\_SlaveTxCpltCallback, the interrput has been disable so there is no STOP EVENT occur regardless Master generate STOP bit.

So there is no way that Slave to detect how many data Master request.

## Receiver:

### Slave\_RxLength < Master\_TxLength:

#### Polling Mode:

Add this block of code

1. **if**(hi2c->XferCount == 1)
2. {
3. CLEAR\_BIT(hi2c->Instance->CR1, I2C\_CR1\_ACK);
4. }

at the end of

1. **while** (hi2c->XferSize > 0U)
2. {
3. ...
4. }

=>

1. **while** (hi2c->XferSize > 0U)
2. {
3. ...
5. **if**(hi2c->XferCount== 1)
6. {
7. CLEAR\_BIT(hi2c->Instance->CR1, I2C\_CR1\_ACK);
8. }
9. }

in this function HAL\_I2C\_Slave\_Receive locating in *stm32f4xx\_hal\_i2c.c* file.

1. HAL\_StatusTypeDef HAL\_I2C\_Slave\_Receive(I2C\_HandleTypeDef \*hi2c, uint8\_t \*pData, uint16\_t Size, uint32\_t Timeout)
2. {
3. ...
4. }

to force the Slave send a NACK to Master when Slave don’t want to receive more data, so that Master will generate STOP bit to stop the conversation.

#### Interrput Mode:

The same idea to Polling Mode, Slave has to send a NACK if it does not want to receive any data from Master, so that Master can stop the conversation by sending STOP bit.

Add this block of code

1. **if**(hi2c->XferCount== 1)
2. {
3. CLEAR\_BIT(hi2c->Instance->CR1, I2C\_CR1\_ACK);
4. }

at the end of

1. **if** (hi2c->XferCount != 0U)
2. {
3. ...
4. }

=>

1. **if** (hi2c->XferCount != 0U)
2. {
3. ...
4. **if**(hi2c->XferCount == 1)
5. {
6. CLEAR\_BIT(hi2c->Instance->CR1, I2C\_CR1\_ACK);
7. }
8. }

in the I2C\_SlaveReceive\_RXNE function locating in *stm32f4xx\_hal\_i2c.c* file.

### Slave\_RxLength > Master\_TxLength:

#### Polling Mode:

HAL\_I2C\_Slave\_Receive return HAL\_ERROR because the Master did not transfer the amount of data which Slave need.

I don’t treat the response as HAL\_ERROR so I’ll change the return value to HAL\_OK.

1. **if** (\_\_HAL\_I2C\_GET\_FLAG(hi2c, I2C\_FLAG\_STOPF) == SET)
2. {
3. ...
4. **return** HAL\_ERROR;
5. }

=>

1. **if** (\_\_HAL\_I2C\_GET\_FLAG(hi2c, I2C\_FLAG\_STOPF) == SET)
2. {
3. ...
4. **return** HAL\_OK;
5. }

The condition stament above belong to I2C\_WaitOnRXNEFlagUntilTimeout function locating in *stm32f4xx\_hal\_i2c.c* file.

The acctual amount of data which Slave receive:  
amount of data = Size - hi2c->XferCount

Size is the parameter of the HAL\_I2C\_Slave\_Receive function.

#### Interrput Mode:

By default, HAL\_I2C\_ERROR\_AF is generated if the amount of data which Slave receive is fewer as expect (Master\_TxLength < Slave\_RxLength). So HAL\_I2C\_SlaveRxCpltCallback could not be called.

In that case, commenting the line of code

1. hi2c->ErrorCode |= HAL\_I2C\_ERROR\_AF;

in this condition stament

1. **if** (hi2c->XferCount != 0U)
2. {
3. /\* Set ErrorCode corresponding to a Non-Acknowledge \*/
4. hi2c->ErrorCode |= HAL\_I2C\_ERROR\_AF;
5. }

=>

1. **if** (hi2c->XferCount != 0U)
2. {
3. /\* Set ErrorCode corresponding to a Non-Acknowledge \*/
4. //  hi2c->ErrorCode |= HAL\_I2C\_ERROR\_AF;
5. }

in the I2C\_Slave\_STOPF function locating in *stm32f4xx\_hal\_i2c.c* file.

So that the HAL\_I2C\_SlaveRxCpltCallback could be call, and HAL\_I2C\_Slave\_Receive\_IT could be recall again right in HAL\_I2C\_SlaveRxCpltCallback.

The acctual amount of data which Slave receive:  
amount of data = Size - hi2c->XferCount

Size is the parameter of the HAL\_I2C\_Slave\_Receive\_IT function.