Optimizing I2C write time for ST25DVxxKC dynamic tags

Introduction

The ST25DVxxKC is the evolution of the ST25DVxxK, proposing strong backward compatibility along with improvements and new features, among which I^2C programming speed increase.

This application note explains how to get the better of this I^2C programming speed increase by optimizing I^2C writes time.

Table 1. Applicable products

Type	Part number
Dual interface EEPROM	ST25DV04KC
	ST25DV16KC
	ST25DV64KC

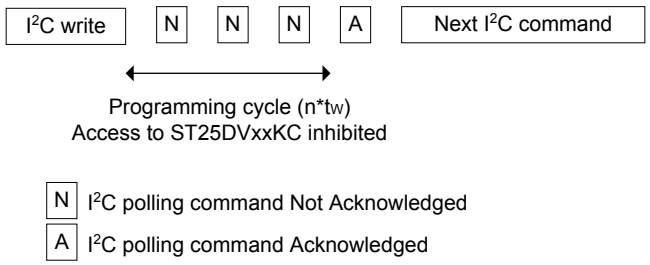
1 I²C programming cycle time

ST25DVxxKC user memory is EEPROM memory type. EEPROM memory technology requires a programming cycle, which consists of an erase phase followed by a programming and a verification phase. This programming cycle requires a certain time to perform all those tasks and writing to an EEPROM memory is not immediate. The EEPROM memory is not available for read or write operation during a programming cycle and data written is only valid at the end of the programming cycle.

When receiving a valid I²C write request in EEPROM memory, the ST25DVxxKC starts the internal programming cycle at the I²C stop condition of the I²C command. During all the programming cycle time, the EEPROM memory access is inhibited and any I²C command sent to the ST25DVxxKC is not acknowledged. This programming cycle time is defined as t_W in the ST25DvxxKC datasheet.

I²C polling usually used to know when the bus is available after a write command (but an alternative method is explained in Section 4 I2C_WRITE GPO interrupt).

Figure 1. I²C write command and programming cycle time



2 Memory organization

ST25DVxxKC internal EEPROM user memory is organized in rows and columns. The organization of the rows and columns is different when addressed from the RF interface and from the I^2C interface.

From the RF interface perspective, the EEPROM user memory is organized as rows of four bytes, and it is only possible to program blocks of four bytes at a time. Those blocks correspond to a memory row.

Table 2. EEPROM user memory as seen from RF interface

RF user memory	0	1	2	3
0	Block 0 Byte 0	Block 0 Byte 1	Block 0 Byte 2	Block 0 Byte 3
1	Block 1 Byte 0	Block 1 Byte 1	Block 1 Byte 2	Block 1 Byte 3
2	Block 2 Byte 0	Block 2 Byte 1	Block 2 Byte 2	Block 2 Byte 3
...	...	...	...	...

For example, as shown in Table 2, block address two corresponds to the third row of the EEPROM user memory.

Programming a memory row of four bytes (one block) from RF interface takes W_{tBLOCK} time, which is around 5 ms depending on temperature conditions (see ST25DVxxKC datasheet for exact W_{tBLOCK} value).

From the I^2C interface perspective, the ST25DVxxKC EEPROM user memory is organized as rows of 16 bytes (versus rows of four bytes in the ST25DVxxK).

Table 3. EEPROM user memory as seen from I^2C interface

I^2C user memory	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
1	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	Byte 30	Byte 31
2	Byte 32	Byte 33	Byte 34	Byte 35	Byte 36	Byte 37	Byte 38	Byte 39	Byte 40	Byte 41	Byte 42	Byte 43	Byte 44	Byte 45	Byte 46	Byte 47
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

For example, as shown in Table 3, Byte address 7, 19 and 32 respectively belong to the first, second and third row of the EEPROM user memory. It is possible to program from 1 byte, up to 256 bytes at a time from the I^2C interface. Nevertheless, whatever the number of bytes to be programmed, the programming is internally done by row and thus the time to program a single byte is the same as the time to program a full row of 16 bytes. This time is defined as t_W in the ST25DVxxKC datasheet (around 5 ms depending on temperature conditions).

3 I²C write optimization

With programming rows of 16 Bytes in 5 ms and the possibility to send I²C write commands of 256 bytes of data, the ST25DVxxKC allows very fast programming of large amount of data.

Nevertheless, programming time can be optimized by making clever use of the EEPROM user memory organization.

As seen in previous chapter, programming time depends on number of rows to be programmed. For instance, programming a single byte, as well as programming several bytes of the same row takes the same time as programming the complete corresponding row.

In the below example, Table 4, the user wants to write 14 Bytes in total in the user memory:

- 1 Byte at memory address 1
- 2 Bytes at memory addresses 19 to 20
- 4 bytes at memory addresses 25 to 28
- 7 Bytes at memory addresses 37 to 43

Table 4. Example of not optimized way of programming less than a row of data

I²C user memory	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-	Byte 1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	Byte 19	Byte 20	-	-	-	-	Byte 25	Byte 26	Byte 27	Byte 28	-	-	-
2	-	-	-	-	-	Byte 37	Byte 38	Byte 39	Byte 40	Byte 41	Byte 42	Byte 43	-	-	-	-
...	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

To write those 14 Bytes, the user issues four I²C write commands (since data presented in a single sequential write command must be contiguous). Each I²C write command requires 5 ms of programming time, even if there is less than a row to program. The total programming time for those 14 Bytes then is $4*t_W \sim 20$ ms.

An optimized way of writing 14 Bytes would have been to organize data so that all data are contiguous (whenever that is possible) and place it in a same memory row as shown in Table 5.

Table 5. Example of optimized way of programming less than a row of data

I²C user memory	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	-	-
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
...	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

With this way of organizing data in memory, the 14 bytes can be programed with a single I²C sequential write command. The programming time is ~5 ms only in that case.

In another example, the user wants to program 28 Bytes. The Bytes are contiguous but are not aligned at the beginning of a row, as shown in Table 6.

Table 6. Example of not optimized programming of several bytes on multiple rows

I²C user memory	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-	-	-	-	-	-	-	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
1	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	Byte 30	Byte 31
2	Byte 32	Byte 33	Byte 34	-	-	-	-	-	-	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

To write those 28 Bytes, the user issues a single I²C sequential write command of 28 Bytes, starting at memory address 7. The data are contiguous but spread over 3 different rows of memory. Each row is programmed in t_W, so the total programming time is $3*t_W$ ~ 15 ms.

An optimized way of writing 28 Bytes would have been to organize data so that it spread over the minimum number of memory rows, as shown in Table 7.

Table 7. Example of optimized programming of several bytes on multiple rows

I²C user memory	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	Byte 30	Byte 31
2	Byte 32	Byte 33	Byte 34	Byte 35	Byte 36	Byte 37	Byte 38	Byte 39	Byte 40	Byte 41	Byte 42	Byte 43	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

With this alignment of data to optimize the number of memory rows used, only two rows need to be programmed in a single I²C sequential write command. The total programming time is $2*t_W$ ~ 10 ms only in that case.

In a general way, the EEPROM programming cycle time can be calculated, depending on I²C write start address and number of bytes to be written (max 256 bytes). Application can use the following C code to calculate the programming time:

ST25DVxxKC programming cycle = t_W*(((start_address + nb_bytes_to_write - 1)>>4) - (start_address>>4) + 1);

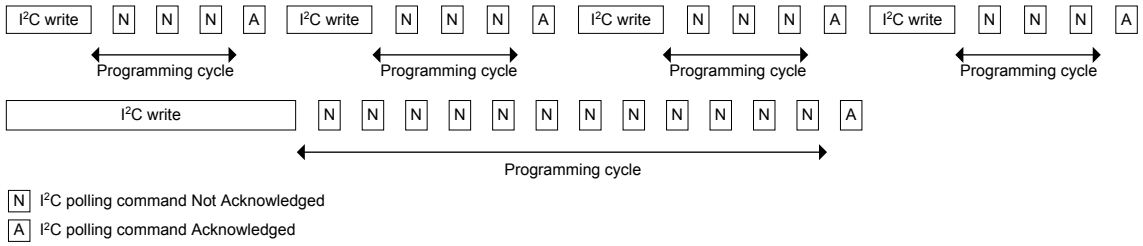
Another way to optimize I²C writing speed, is to group data to write in a minimum number of I²C commands. The ST25DVxxKC can treat I²C sequential write commands of up to 256 Bytes. The device internally buffers the data received from the I²C write command and starts to program the memory row by row in an optimal timing after the stop condition.

It is more efficient to send only one long I²C sequential write command than several small I²C write commands for several reasons:

- Optimized usage of the memory rows programming time, as seen previously.
- Less I²C protocol overhead: for each I²C write command, 3 Bytes must be sent before the data: one Byte for device address and two Bytes for memory address.
- Less time lost between I²C write commands: this is the time taken by the I²C master to prepare the next I²C command.
- Less time lost in polling to check the end of the programming cycle (the next chapter show how this can be optimized): after each I²C write command, the master usually polls the I²C bus to know when it is available for the next command.

The next Figure 2 illustrates the gain of time using a long I²C sequential write command rather than several short I²C write commands. For the same number of data written, the long I²C sequential write command is shorter than the four short commands by 87 I²C clock cycles. The sum of programming cycle time for the four short I²C command is at best equal to the programming cycle of the long I²C command (depending on data organization in memory). The total number of polling commands and time spent by the I²C master to prepare next command also is reduced.

Figure 2. Short I²C write versus long I²C sequential write



As a summary, if writing performances is important to the application, it is recommended to:

- Organize data in memory to minimize the number of I²C writes commands.
- Use I²C sequential write commands as longest as possible (which implies contiguous data organization).
- Align I²C writes addresses on memory rows to minimize the number of rows used. This means aligning with addresses with the 4 less significant bits equal to 0000b.

4 I2C_WRITE GPO interrupt

To reduce the inconvenience of I^2C polling after an I^2C write command, the ST25DVxxKC provides an I2C_WRITE interrupt on the GPO pin at the completion of the I^2C EEPROM programming cycle. This interrupt can be advantageously used by the I^2C master to know exactly when the ST25DVxxKC is ready to receive a new I^2C command after an I^2C write.

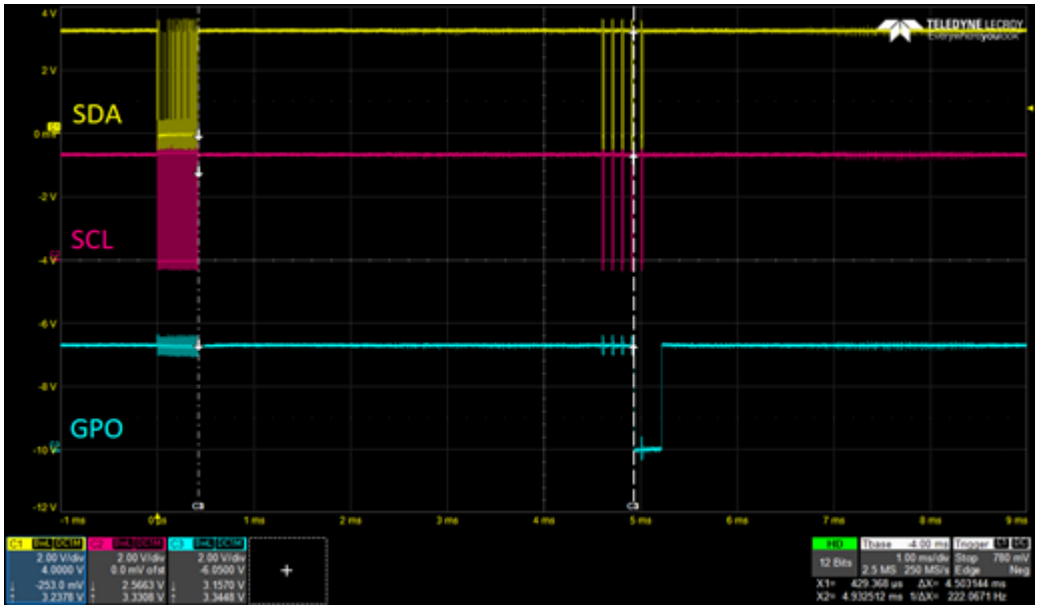
The benefits of using this interrupt instead of polling the I^2C bus are double:

- reduce lost time due to polling delay
- reduce CPU load of for I2C master

The I2C_WRITE interrupt is triggered on the GPO pin at the completion of the programming cycle. As soon as the GPO line goes low (for open drain version, or high for CMOS version), the ST25DVxxKC is ready to receive a new I^2C command.

Next Figure 3 shows an example of I2C_WRITE interrupt. In this example, an I^2C write command is received by the ST25DVxxKC (yellow and pink traces for SDA and SCL signals). The GPO pin (blue trace) goes low as soon as the EEPROM programing cycle is finished. Some I^2C polling commands are also present but are only shown as reference for the example and are not necessary: the first four polling commands in this example are not acknowledged, and only the fifth one, which occurs after the GPO goes low, is acknowledged.

Figure 3. GPO I2C_WRITE interrupt versus I^2C polling



Revision history

Table 8. **Document revision history**

Date	Version	Changes
17-Feb-2022	1	Initial release.

Contents

List of tables

List of figures