Page program operation with buffer load usage

Introduction

This document provides guidelines on the usage of the page program operation for the products listed in Table 1. These are page EEPROM (electrically erasable programmable read only memory) devices manufactured with an advanced NVM technology, offering byte flexibility, page alterability, high page cycling performance, and ultra low power consumption.

When a significant part of the memory content must be programmed, the speed is critical for manufacturing costs and downtimes during the application lifetime. This can be achieved using the page program with buffer load operation, which significantly speeds up the device programming.

Table 1. Applicable products

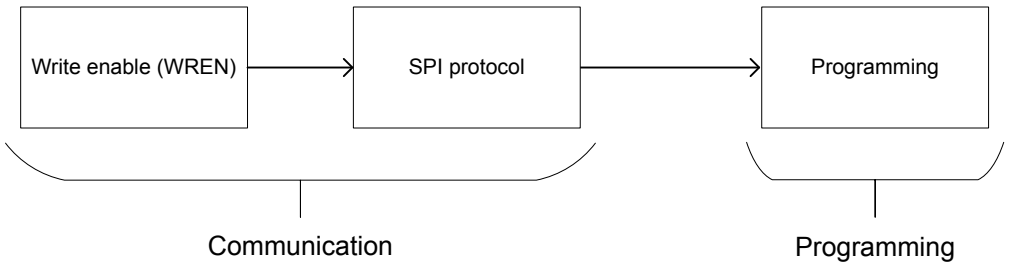
Series	Root part number
Standard Serial Page EEPROM	M95P32-E M95P32-I
	M95P16-E M95P16-I
	M95P08-E M95P08-I

1 Page program with buffer load definition

1.1 Page program with buffer load operation

This operation is used to program from one to 512 bytes of data initially in the erased state (0xFF). The command must be sent through the SPI bus from the master to the device. When the command is received, the device starts the actual programming, as shown in Figure 1.

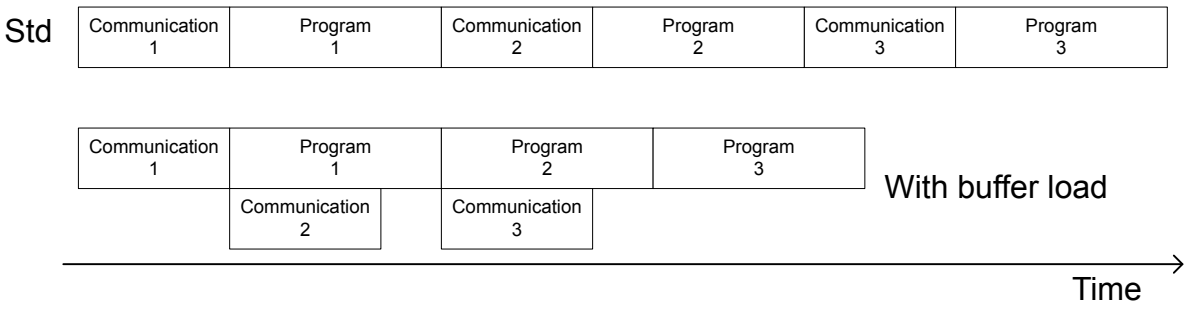
Figure 1. Page program operation



To program more than one page, successive page program instructions must be sent. When a significant part of the memory must be programmed, the page program with buffer load operation can reduce the communication overhead.

In Figure 2 there is a comparison of the page program operations (standard on top, with buffer load on bottom).

Figure 2. Regular 3x page program operations



During the standard operation the communication takes place before the programming (these steps are in sequence). The WREN instruction is executed for each programming step, and the overall time for programming three pages is 3x communication time + 3x programming time.

When the buffer load is used, the communication steps (except the first one) are executed at the same time as the programming steps. The WREN instruction is executed only once (at the beginning of the whole programming) and the overall time to program three pages is 1x communication time + 3x programming time.

Programming pages with buffer load is therefore faster. To activate the buffer load, the BUFEN bit in the volatile register must be set to "1".

Table 2. Volatile register format

b7	b6	b5	b4	b3	b2	b1	b0
X	X	X	X	X	X	BUFEN	BUFLD

1.2 Page program and communication timing

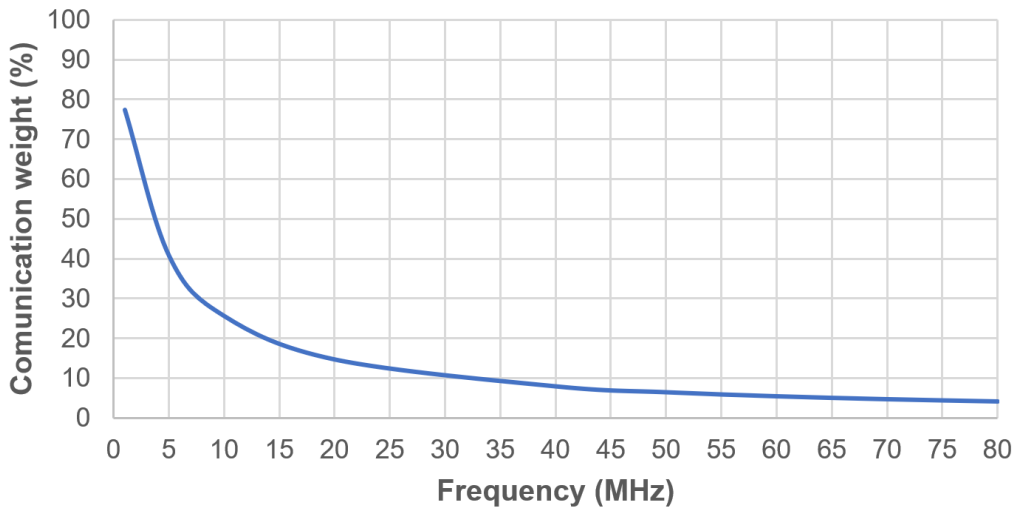
Page program timing for M95P32 is typically 1.2 ms over the SPI bus frequency range. However, the time taken by the communication is frequency-dependent, as shown in Table 3.

Table 3. Estimated communication time for a page program operation (512 bytes)

Frequency	Communication time
1 MHz	4.0 ms
5 MHz	0.8 ms
10 MHz	0.4 ms
20 MHz	0.2 ms
40 MHz	0.1 ms
50 MHz	0.08 ms
60 MHz	0.07 ms
70 MHz	0.06 ms
80 MHz	0.05 ms

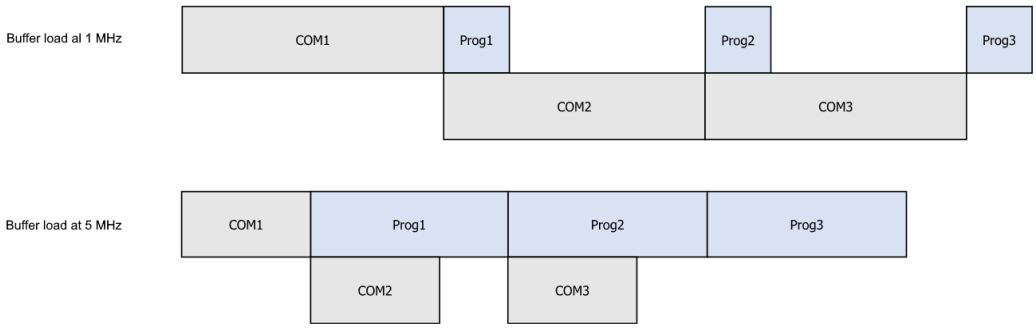
With the typical programming time value, it is possible to determine the weight of the communication in a page program operation. Figure 3 details the percentage of the total programming time taken by the communication in a page program operation, versus the SPI bus frequency.

Figure 3. Communication weight for a typical page program operation (512 bytes)



When the SPI bus frequency is 80 MHz, the communication takes 4% of the total time, this value jumps to 77% when the frequency is 1 MHz. Below 4 MHz the communication takes longer than the programming itself. To maximize the device performance, it is recommended to use the buffer load feature above 4 MHz.

Figure 4. Page program with buffer load



As illustrated in Figure 4, at 1 MHz the communication is longer than the programming, only part of it is executed during the programming step. At 5 MHz, the communication uses 40% of the page program operation time, hence it is executed during the programming step.

Application example

An application based on M95P32 needs to program 4096 pages (half of the memory size). The application operates at 12.5 MHz.

The page program operation consists of three parts:

- WREN instruction: 0.64 µs at 12.5 MHz
- Page program instruction: 330 µs at 12.5 MHz
- 1 page (512 bytes) programming: 1200 µs

Page program without buffer load: the overall time to program 4096 pages at 12 MHz is (330 + 1200 + 0.64) x 4096 = 6.27 s, hence the communication (1.35 s) uses 20% of the total timing.

Page program with buffer load: the overall time to program 4096 pages at 12 MHz is (4096 x 1200) + 0.64 + 330 = 4.9 s, hence the communication (0.00033 s) takes 0.007% of the total timing.

In this example, the page program with buffer load is 1.3 times faster than the regular page program operation. The larger the amount of data to program, the more the page program with buffer load is efficient.

Below 3.5 MHz the communication time is greater than the programming time. It is then recommended to use the page program with buffer load operation at higher frequencies. For instance, at 12.5 MHz, programming 4096 pages with the buffer load is 30% faster when compared with the standard page program operation. As indicated in Section 1.1 , this feature is activated setting BUFEN bit to 1.

2 Buffer load algorithm

This algorithm is available in the STM32 Nucleo expansion boards dedicated to the page EEPROM products.

2.1 Buffer load prerequisites

To start a page program operation with buffer load, it is recommended to:

- know page and block sizes and the number of blocks of the memory
- determine start and end addresses of area that must be initially erased
- check if targeted area is erased and not protected

The buffer load algorithm guideline is illustrated for the M95P32.

3 Buffer load steps

The input data for the buffer load steps are:

- `start_prog_add`: address of the first byte to be programmed
- `byte_array[]`: array to be programmed

This algorithm is composed of these steps:

1. Determine the number of bytes to program

The size of the `byte_array` gives the number of bytes to be programmed.

```
sizeof (byte_array) = nb_byte
```

2. Erase check

ST page EEPROM devices are organized in pages of 512 bytes. To program a byte inside a word, the entire word must be erased. Therefore, if the start address is 0x05 the byte from address 0x00 to address 0x0F must be erased (for more information, refer to the application note AN5747).

The purpose of this step is to ensure that the requested programming area is erased. A start and an end address define this area.

Start_erase_add calculation

With the ECC management, the `start_erase_add` must be the first byte of the `start_prog_add` word. Here a code example to determine the `start_erase_add`:

- `start_prog_add%16 = result`
- If `result = 0 then start_erase_add = start_prog_add`
- Else `start_erase_add = start_prog_add – result`

End_add calculation

With the ECC management, the `end_erase_add` must be the last byte of the `end_prog_add` word. Here a code example to determine the `end_erase_add`:

- `end_prog_add = start_prog_add + nb_byte -1`
- `end_prog_add%15 = result`
- If `result = 0 then end_erase_add = end_prog_add`
- Else `end_erase_add = end_prog_add + (16 - result)`

3. Check if area is protected

- READ status register to check TB and BP bits.
  Page EEPROM products configure the memory area as read-only to protect sensitive data. Page program over protected data are discarded. Therefore, before programming, the algorithm must not attempt to program protected data.

Table 4. Status register format

b7	b6	b5	b4	b3	b2	b1	b0
SRWD	TB	X	BP2	BP1	BP0	WEL	WIP

SR = status register reading result
`BP_bits = (SR & 0x1C) >> 2`

Table 5. BP and TB bits (M95P32 only)

BP_bits	Protection	Action
0	No protection	Next step
7	Full protection	Algorithm aborted
0 < BP_bits < 7 TB_bit = 0	Protected from the top	Determine *bot_add_secure*
0 < BP_bits < 7 TB_bit = 1	Protected from the bottom	Determine *top_add_secure*

The goal is to determine which part of the memory is protected and if the algorithm is going to program this part. Only one address has to be calculated. The memory is protected from the top to the bottom address (TB_bit = 0) or from the bottom to the top address (TB_bit = 1).
Here a code example to determine top or bottom secure addresses:
Determine `top_add_secure`

– `top_add_secure = (block_size x 2 (BP_bits- 1)) -1`
– if `start_erase_add ≤ top_add_secure`, algorithm aborted because area to program is protected, else next step.

Determine `bot_add_secure`

– `bot_add_secure = block_size x (block_nb -2 (BP_bits - 1))`
– If `end_erase_add ≥ bot_add_secure`, algorithm aborted because area to program is protected, else next step.
  Example:
  `Status register = 0x50` and `start_erase_add =0x300000`
  `TB_bit = 1 BP2 = 1 BP1 = 0 BP0 = 0`

Area is protected from the bottom `top_add_secure` must be calculated:
`top_add_secure = 65536x 2^3 = 5 242 87= 0x7FFFF`
`Start_erase_add > top_add_secure`
protection covers 0x0000–0x7FFFF addresses. Programing area is not protected, next step.

4. Check if the area is erased

The area to program must be erased. Any page program operation over data not in erased state is discarded. This step must check if the area to be programmed has been erased without error.

- `READ` operation from `start_erase_add to end_erase_add`
- `READ` safety register to check ECC bits, if error(s) then algorithm aborted
- If read data ≠ 0xFF then algorithm aborted, else next step.

Table 6. Safety register format

b7	b6	b5	b4	b3	b2	b1	b0
PAMAF	PUF	ERF	PRF	ECC1C	ECC2C	ECC3D	ECC3DS

5. Check start and end address page alignment

Page program with buffer load programs `n` bytes starting at `start_prog_add`. If `start_prog_add` is in the middle of a page, the algorithm must prevent the page rollover. Moreover, if `end_prog_add` is also in the middle of a page, the algorithm must prevent programming bytes located after `end_prog_add`.

- If `start_prog_add%512!=0` then the first page program is managed outside of an array.
- If `end_prog_add%511!=0` then the last page program is managed outside of an array.

6. Setup buffer load

- Write enable
- Write volatile register (see Figure 5) with value 0x02 to set BUFFEN bit to 1

7. Program in buffer mode

- Write enable only once
- Loop to program (page program) all the pages in buffer mode from `start_prog_add` to `end_prog_add`:
  - Page program
  - READ volatile register (see Figure 5) to check buffer loading status (`BUFLD`) bit
  - If `BUFLD = 0` the next page program is sent, else polling until `BUFLD = 0`

  The master reads the volatile register to check `BUFLD` when `BUFLD = 0`, a new page instruction is sent.

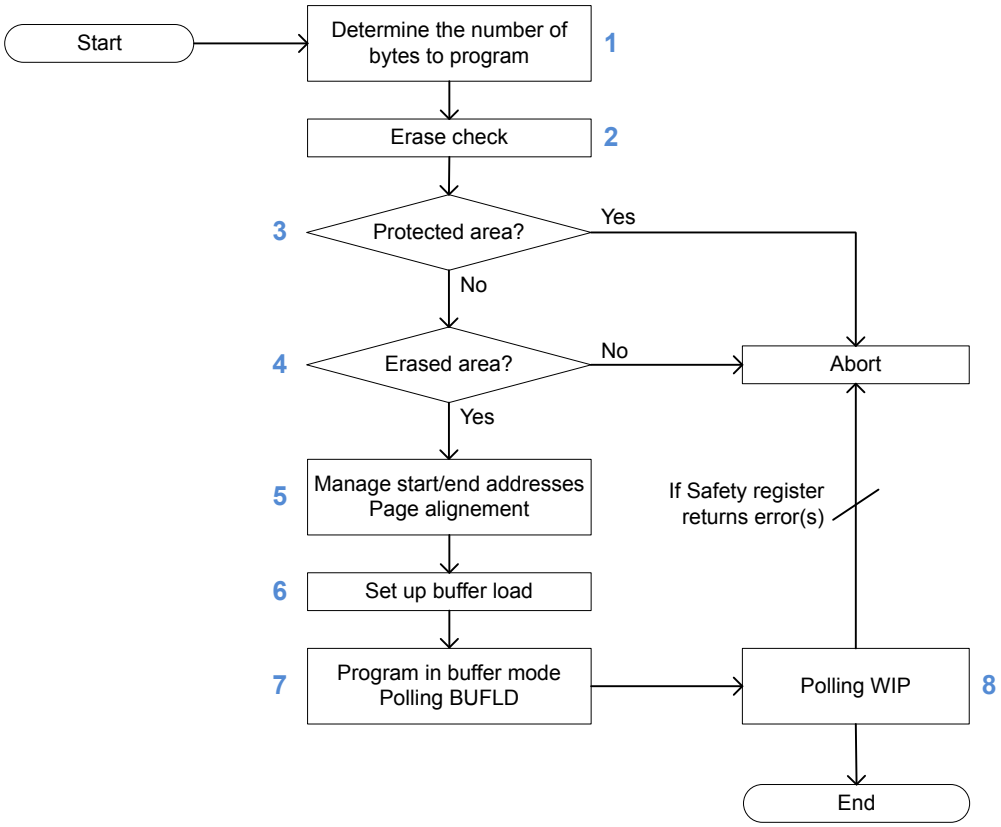
8. Final polling

Once the last page program is sent, polling checks the WIP bit (see Figure 5):

- READ status register to check WIP bit until WIP = 0
- READ safety register to check if a flag (error during page program) is set, if error then algorithm aborted
- Write enable + write volatile register (see Figure 5) with value 0x01 to set BUFFEN bit to 0. Mandatory to read programmed data
- READ volatile register to check BUFFEN bit = 0
- Check programmed data with a READ from `start_prog_add` to `end_prog_add`
- READ safety register (see Figure 5) to check ECC bits

4 Buffer load algorithm

Figure 5. Buffer load algorithm



Revision history

Table 7. **Document revision history**

Date	Version	Changes
13-Jun-2022	1	Initial release.
25-Jul-2022	2	Document scope limited to M95P32-E and M95P32-I, hence updated Section Introduction.
23-Feb-2023	3	Document scope extended to M95P08-E, M95P08-I, M95P16-E, and M95P16-I devices, hence added Table 1. Applicable products.

Contents

List of tables

List of figures