SSD1306

Advance Information

128 x 64 Dot Matrix
OLED/PLED Segment/Common Driver with Controller

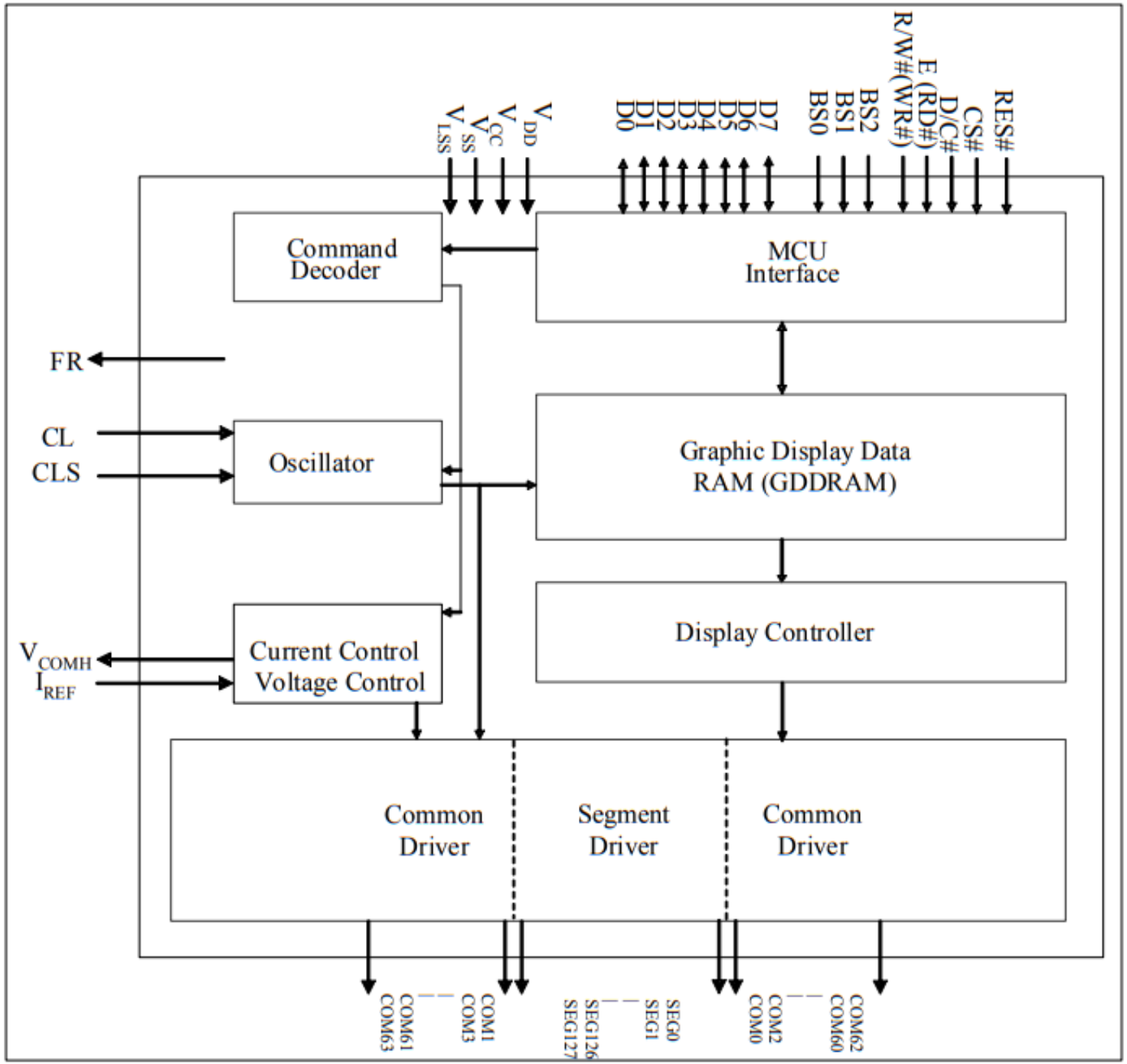


Figure 4-1 SSD1306 Block Diagram

8 FUNCTIONAL BLOCK DESCRIPTIONS

8.1 MCU Interface selection

SSD1306 MCU interface consist of 8 data pins and 5 control pins. The pin assignment at different interface mode is summarized in Table 8-1. Different MCU mode can be set by hardware selection on BS[2:0] pins (please refer to Table 7-1 for BS[2:0] setting).

Table 8-1 : MCU interface assignment under different bus interface mode

Pin Name / Bus Interface	Data/Command Interface								Control Signal				
	D7	D6	D5	D4	D3	D2	D1	D0	E	R/W#	CS#	D/C#	RES#
8-bit 8080	D[7:0]								RD#	WR#	CS#	D/C#	RES#
8-bit 6800	D[7:0]								E	R/W#	CS#	D/C#	RES#
3-wire SPI	Tie LOW					NC	SDIN	SCLK	Tie LOW		CS#	Tie LOW	RES#
4-wire SPI	Tie LOW					NC	SDIN	SCLK	Tie LOW		CS#	D/C#	RES#
I^2C	Tie LOW					SDA_{OUT}	SDA_{IN}	SCL	Tie LOW			SA0	RES#

8.1.5 MCU I^2C Interface

The I^2C communication interface consists of slave address bit SA0, I^2C-bus data signal SDA (SDA_{OUT}/D_2 for output and SDA_{IN}/D_1 for input) and I^2C-bus clock signal SCL (D_0). Both the data and clock signals must be connected to pull-up resistors. RES# is used for the initialization of device.

a) Slave address bit (SA0)
SSD1306 has to recognize the slave address before transmitting or receiving any information by the I^2C-bus. The device will respond to the slave address following by the slave address bit ("SA0" bit) and the read/write select bit ("R/W#" bit) with the following byte format,

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	1	1	1	1	0	SA0	R/W#

"SA0" bit provides an extension bit for the slave address. Either "0111100" or "0111101", can be selected as the slave address of SSD1306. D/C# pin acts as SA0 for slave address selection.
"R/W#" bit is used to determine the operation mode of the I^2C-bus interface. R/W#=1, it is in read mode. R/W#=0, it is in write mode.

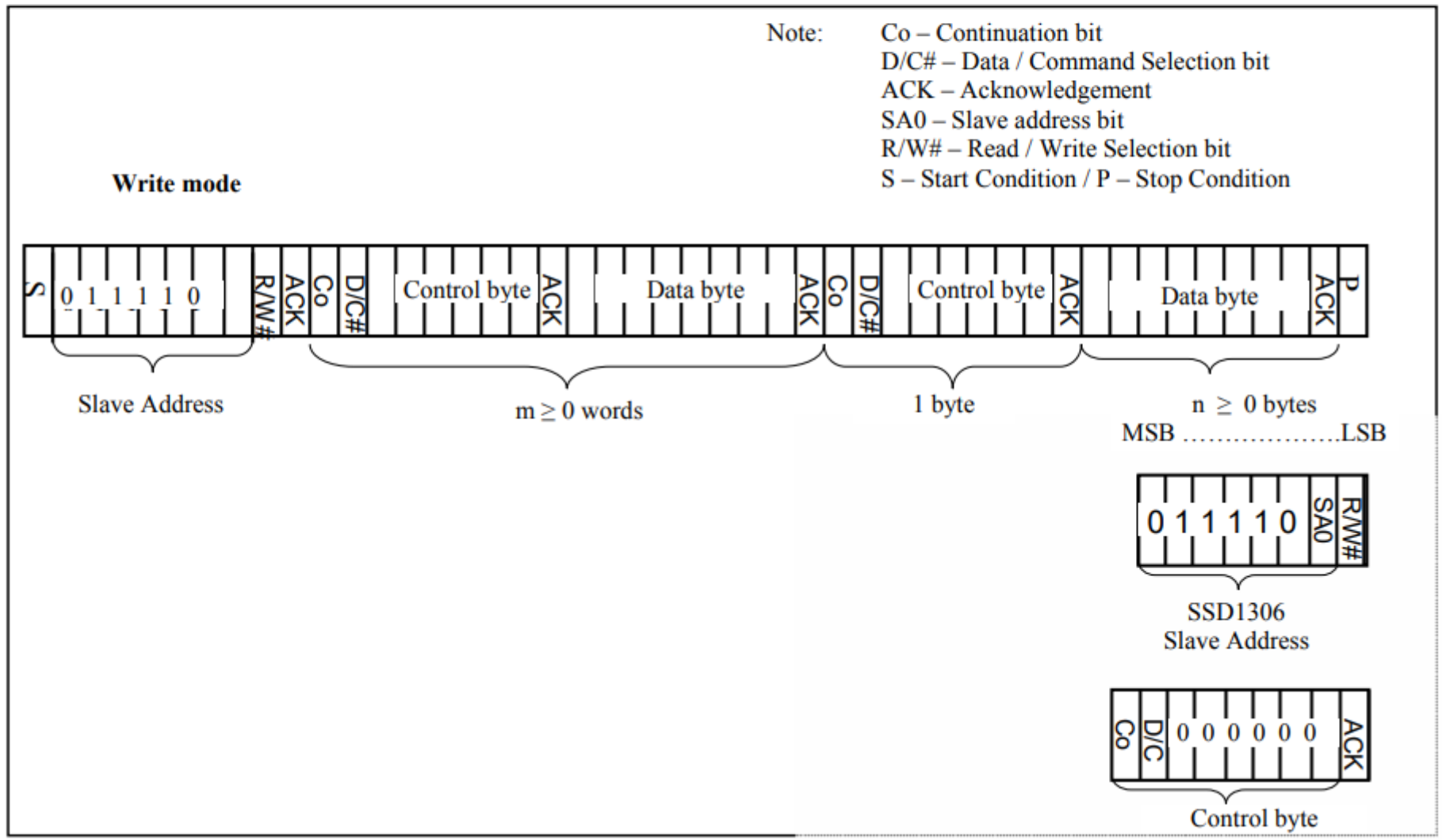
b) I^2C-bus data signal (SDA)
SDA acts as a communication channel between the transmitter and the receiver. The data and the acknowledgement are sent through the SDA.
It should be noticed that the ITO track resistance and the pulled-up resistance at "SDA" pin becomes a voltage potential divider. As a result, the acknowledgement would not be possible to attain a valid logic 0 level in "SDA".
"SDA_{IN}" and "SDA_{OUT}" are tied together and serve as SDA. The "SDA_{IN}" pin must be connected to act as SDA. The "SDA_{OUT}" pin may be disconnected. When "SDA_{OUT}" pin is disconnected, the acknowledgement signal will be ignored in the I^2C-bus.

c) I^2C-bus clock signal (SCL)
The transmission of information in the I^2C-bus is following a clock signal, SCL. Each transmission of data bit is taken place during a single clock period of SCL.

8.1.5.1 I²C-bus Write data

The I²C-bus interface gives access to write data and command into the device. Please refer to Figure 8-7 for the write mode of I²C-bus in chronological order.

Figure 8-7 : I²C-bus data format



8.1.5.2 Write mode for I^2C

1) The master device initiates the data communication by a start condition. The definition of the start condition is shown in Figure 8-8. The start condition is established by pulling the SDA from HIGH to LOW while the SCL stays HIGH.
2) The slave address is following the start condition for recognition use. For the SSD1306, the slave address is either "b0111100" or "b0111101" by changing the SA0 to LOW or HIGH (D/C pin acts as SA0).
3) The write mode is established by setting the R/W# bit to logic "0".
4) An acknowledgement signal will be generated after receiving one byte of data, including the slave address and the R/W# bit. Please refer to the Figure 8-9 for the graphical representation of the acknowledge signal. The acknowledge bit is defined as the SDA line is pulled down during the HIGH period of the acknowledgement related clock pulse.
5) After the transmission of the slave address, either the control byte or the data byte may be sent across the SDA. A control byte mainly consists of Co and D/C# bits following by six "0" 's.
    a. If the Co bit is set as logic "0", the transmission of the following information will contain data bytes only.
    b. The D/C# bit determines the next data byte is acted as a command or a data. If the D/C# bit is set to logic "0", it defines the following data byte as a command. If the D/C# bit is set to logic "1", it defines the following data byte as a data which will be stored at the GDDRAM. The GDDRAM column address pointer will be increased by one automatically after each data write.
6) Acknowledge bit will be generated after receiving each control byte or data byte.
7) The write mode will be finished when a stop condition is applied. The stop condition is also defined in Figure 8-8. The stop condition is established by pulling the "SDA in" from LOW to HIGH while the "SCL" stays HIGH.

Figure 8-8 : Definition of the Start and Stop Condition

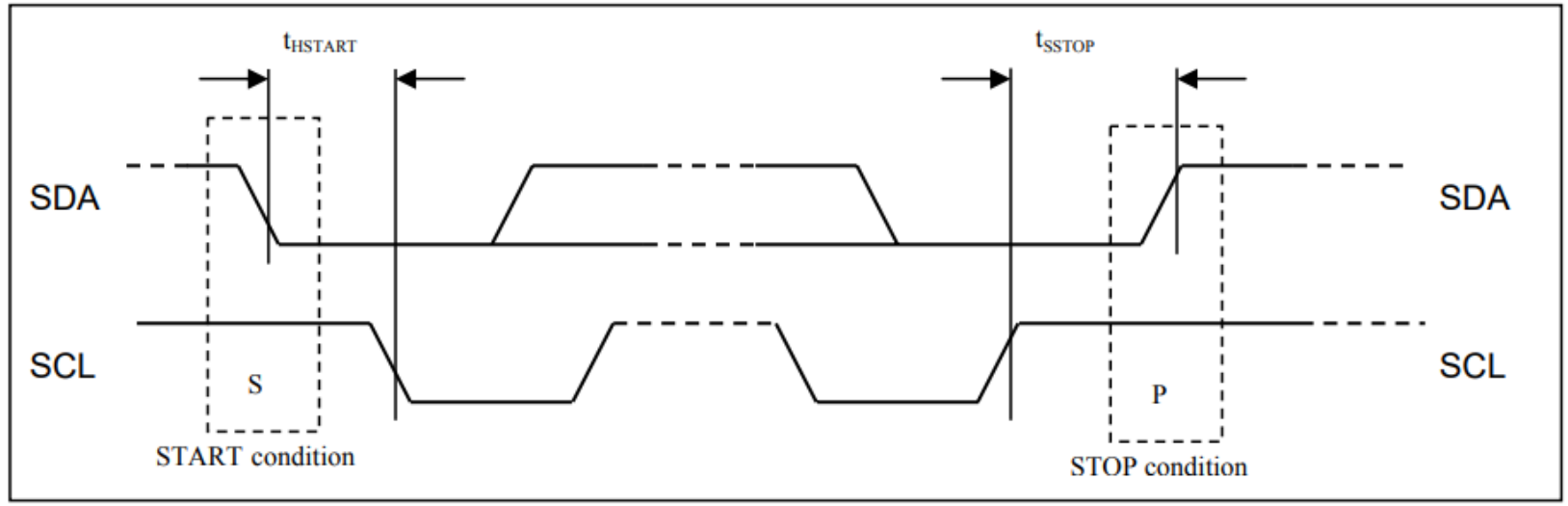
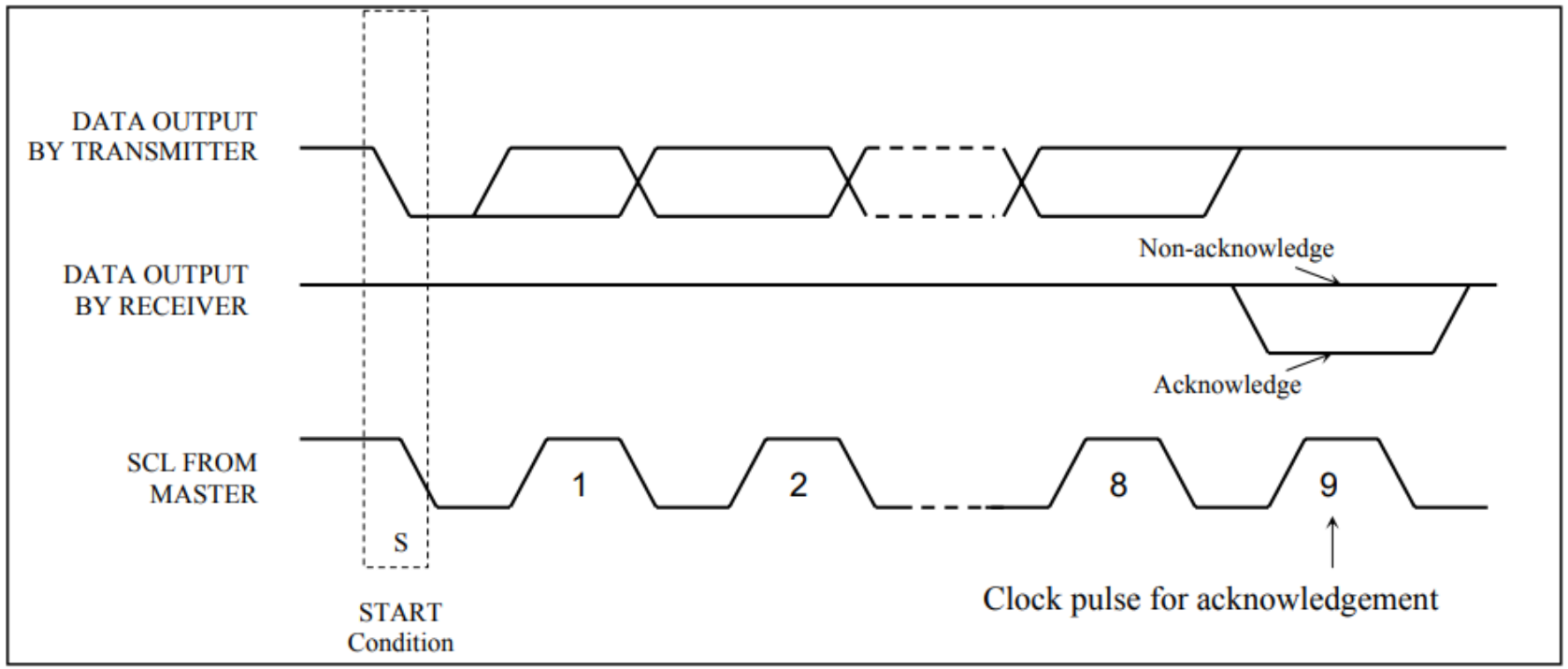


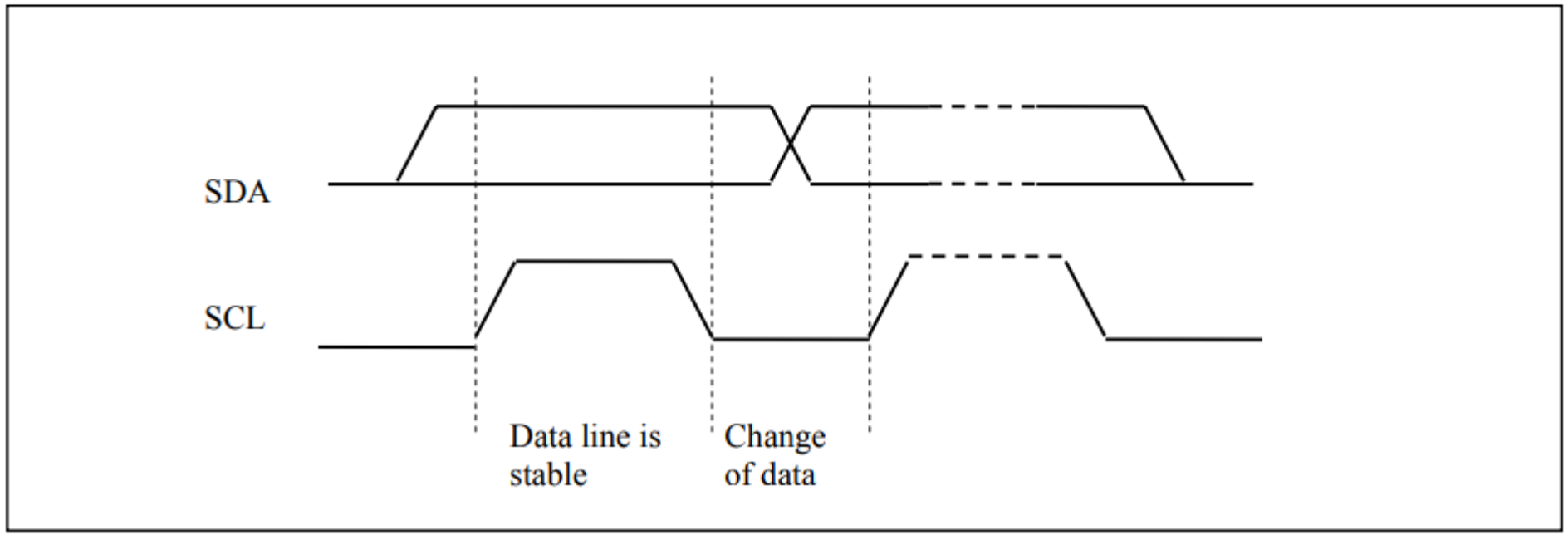
Figure 8-9 : Definition of the acknowledgement condition



Please be noted that the transmission of the data bit has some limitations.

1. The data bit, which is transmitted during each SCL pulse, must keep at a stable state within the "HIGH" period of the clock pulse. Please refer to the Figure 8-10 for graphical representations. Except in start or stop conditions, the data line can be switched only when the SCL is LOW.
2. Both the data line (SDA) and the clock line (SCL) should be pulled up by external resistors.

Figure 8-10 : Definition of the data transfer condition



8.5 Reset Circuit

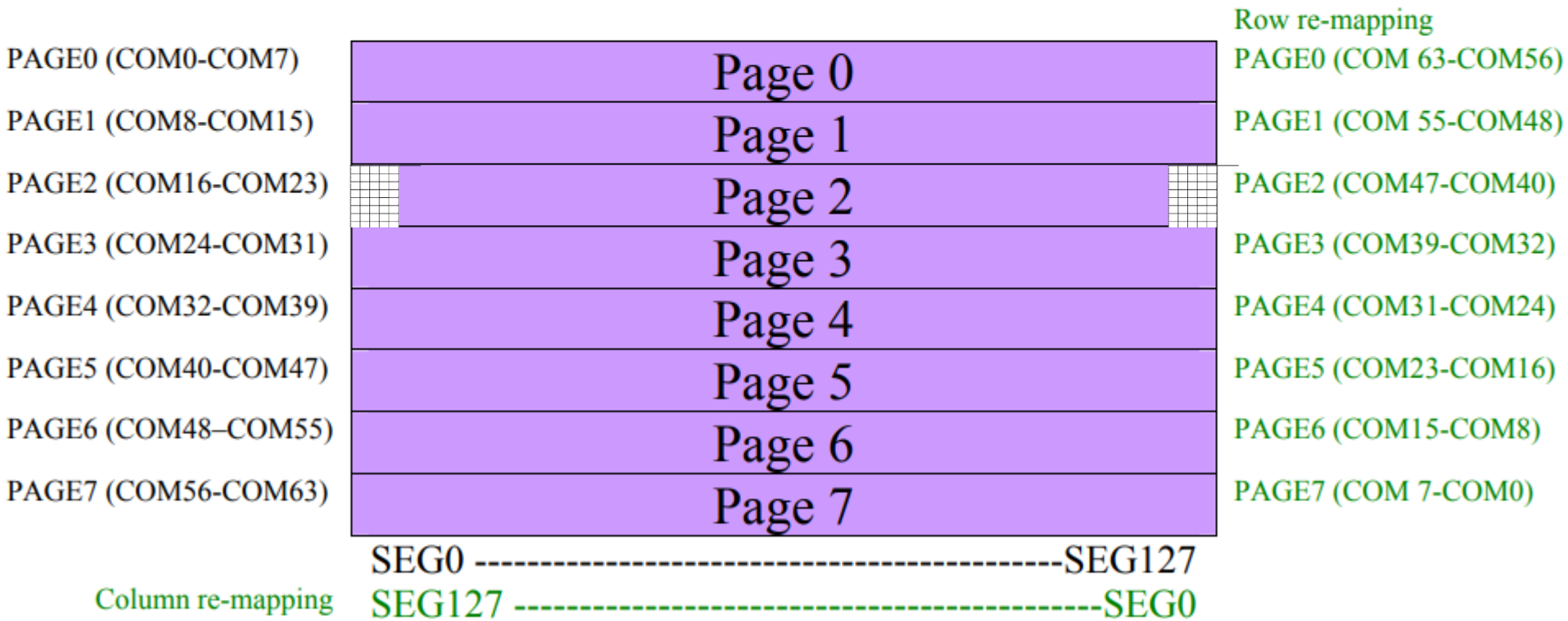
When RES# input is LOW, the chip is initialized with the following status:

1. Display is OFF
2. 128 x 64 Display Mode
3. Normal segment and display data column address and row address mapping (SEG0 mapped to address 00h and COM0 mapped to address 00h)
4. Shift register data clear in serial interface
5. Display start line is set at display RAM address 0
6. Column address counter is set at 0
7. Normal scan direction of the COM outputs
8. Contrast control register is set at 7Fh
9. Normal display mode (Equivalent to A4h command)

8.7 Graphic Display Data RAM (GDDRAM)

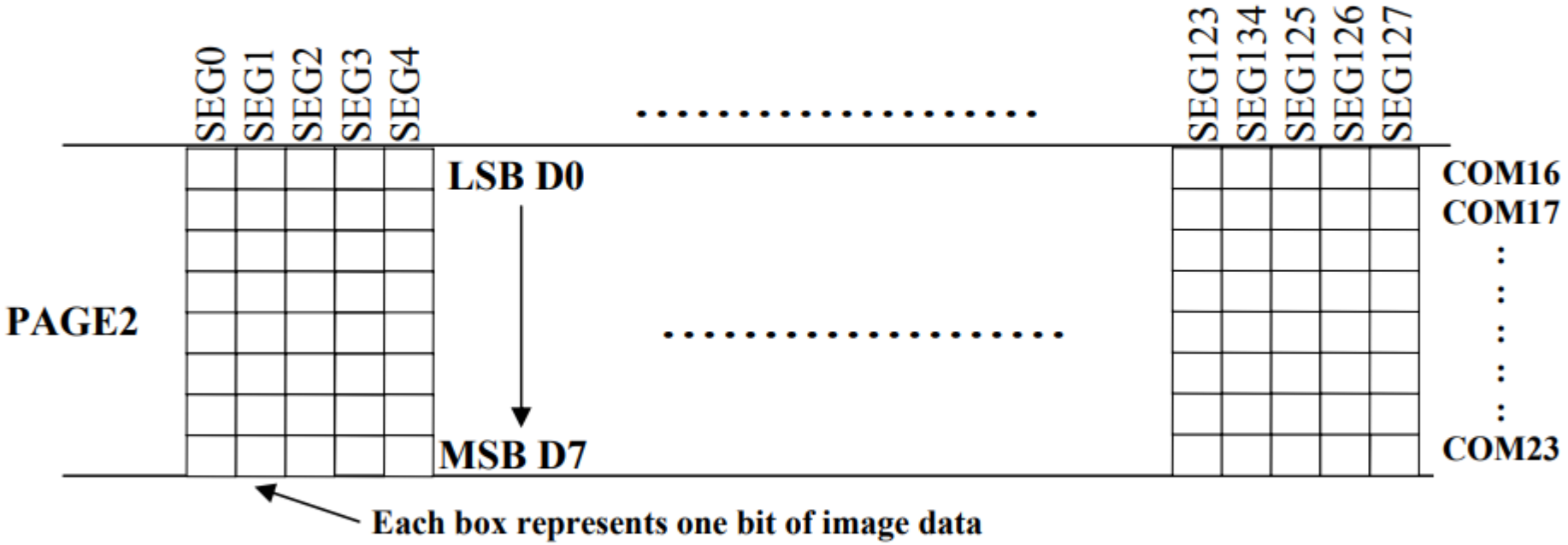
The GDDRAM is a bit mapped static RAM holding the bit pattern to be displayed. The size of the RAM is 128 x 64 bits and the RAM is divided into eight pages, from PAGE0 to PAGE7, which are used for monochrome 128x64 dot matrix display, as shown in Figure 8-13.

Figure 8-13 : GDDRAM pages structure of SSD1306



When one data byte is written into GDDRAM, all the rows image data of the same page of the current column are filled (i.e. the whole column (8 bits) pointed by the column address pointer is filled.). Data bit D0 is written into the top row, while data bit D7 is written into bottom row as shown in Figure 8-14.

Figure 8-14 : Enlargement of GDDRAM (No row re-mapping and column-remapping)



For mechanical flexibility, re-mapping on both Segment and Common outputs can be selected by software as shown in Figure 8-13.

For vertical shifting of the display, an internal register storing the display start line can be set to control the portion of the RAM data to be mapped to the display (command D3h).

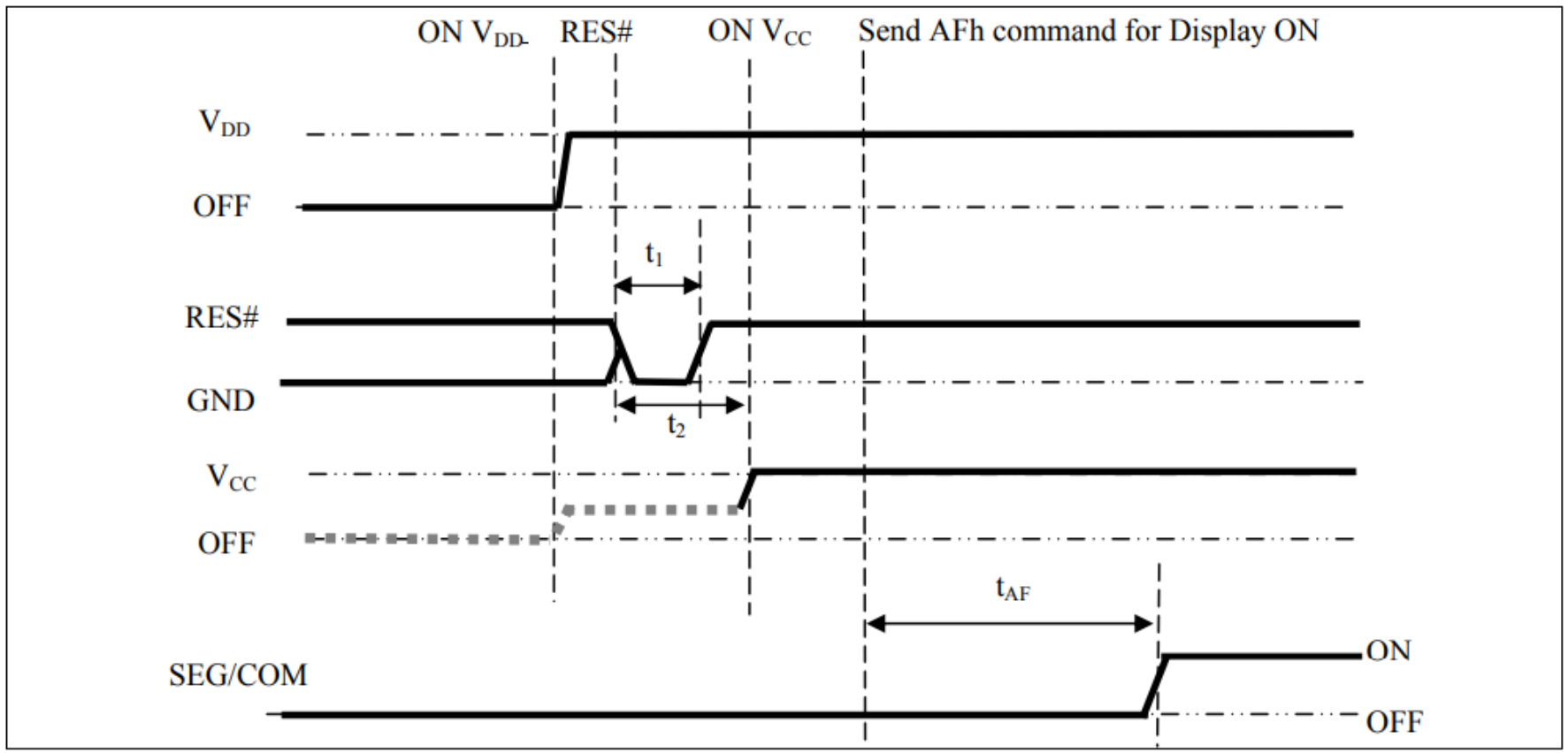
8.9 Power ON and OFF sequence

The following figures illustrate the recommended power ON and power OFF sequence of SSD1306

Power ON sequence:

1. Power ON V_{DD}
2. After V_{DD} become stable, set RES# pin LOW (logic low) for at least 3us (t_1) [4] and then HIGH (logic high).
3. After set RES# pin LOW (logic low), wait for at least 3us (t_2). Then Power ON V_{CC}. [1]
4. After V_{CC} become stable, send command AFh for display ON. SEG/COM will be ON after 100ms (t_{AF}).

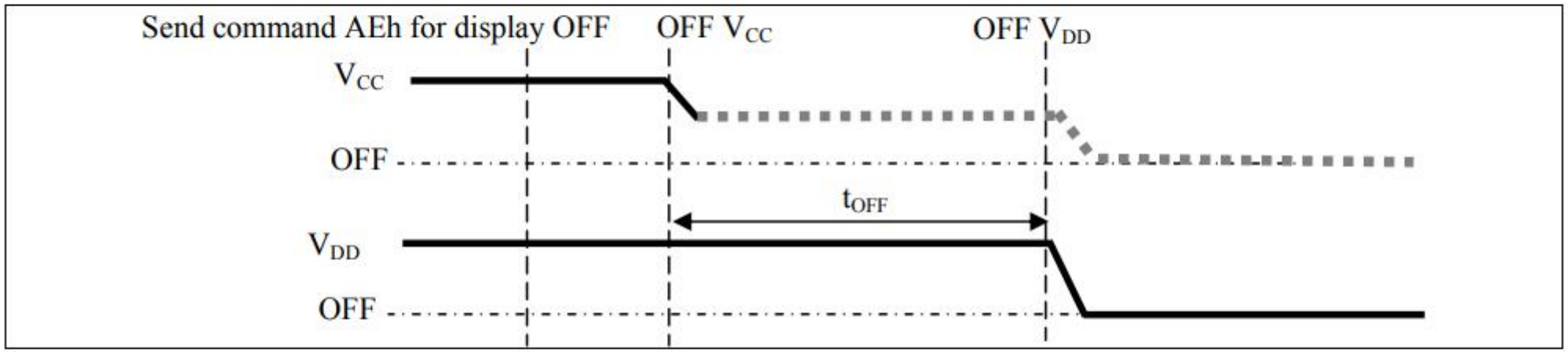
Figure 8-16 : The Power ON sequence



Power OFF sequence:

1. Send command AEh for display OFF.
2. Power OFF V_{CC}.[(1), (2), (3)]
3. Power OFF V_{DD} after t_{OFF}.[(5)] (Typical t_{OFF}=100ms)

Figure 8-17 : The Power OFF sequence



Note:

(1) Since an ESD protection circuit is connected between V_{DD} and V_{CC}, V_{CC} becomes lower than V_{DD} whenever V_{DD} is ON and V_{CC} is OFF as shown in the dotted line of V_{CC} in Figure 8-16 and Figure 8-17.

(2) V_{CC} should be kept float (i.e. disable) when it is OFF.

(3) Power Pins (V_{DD} , V_{CC}) can never be pulled to ground under any circumstance.

(4) The register values are reset after t_1.

(5) V_{DD} should not be Power OFF before V_{CC} Power OFF.

9 COMMAND TABLE

Table 9-1: Command Table

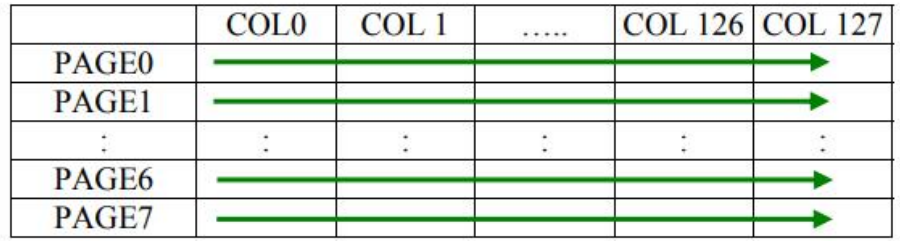
(D/C#=0, R/W#(WR#) = 0, E(RD#=1) unless specific setting is stated)

1. Fundamental Command Table											
D/C#	**Hex**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Command**	**Description**
0 0	81 A[7:0]	1 A_7	0 A_6	0 A_5	0 A_4	0 A_3	0 A_2	0 A_1	1 A_0	Set Contrast Control	Double byte command to select 1 out of 256 contrast steps. Contrast increases as the value increases. (RESET = 7Fh)
0	A4/A5	1	0	1	0	0	1	0	X_0	Entire Display ON	A4h, X_0=0b: Resume to RAM content display (RESET) Output follows RAM content A5h, X_0=1b: Entire display ON Output ignores RAM content
0	A6/A7	1	0	1	0	0	1	1	X_0	Set Normal/Inverse Display	A6h, X[0]=0b: Normal display (RESET) 0 in RAM: OFF in display panel 1 in RAM: ON in display panel A7h, X[0]=1b: Inverse display 0 in RAM: ON in display panel 1 in RAM: OFF in display panel
0	AE AF	1	0	1	0	1	1	1	X_0	Set Display ON/OFF	AEh, X[0]=0b:Display OFF (sleep mode) (RESET) AFh X[0]=1b:Display ON in normal mode

Page addressing mode (A[1:0]=10xb)

In page addressing mode, after the display RAM is read/written, the column address pointer is increased automatically by 1. If the column address pointer reaches column end address, the column address pointer is reset to column start address and page address pointer is not changed. Users have to set the new page and column addresses in order to access the next page RAM content. The sequence of movement of the PAGE and column address point for page addressing mode is shown in Figure 10-1.

Figure 10-1 : Address Pointer Movement of Page addressing mode

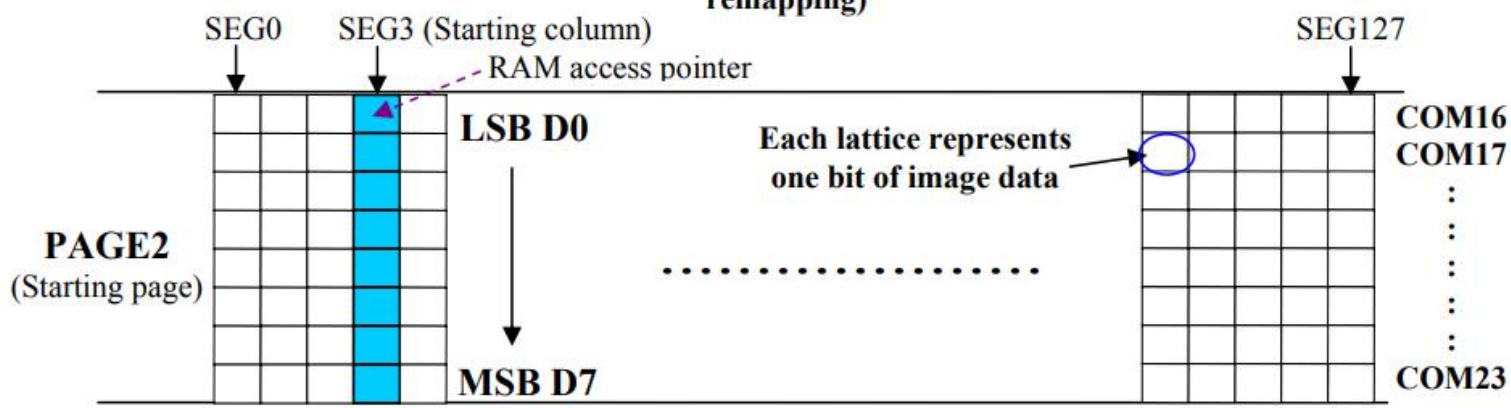


In normal display data RAM read or write and page addressing mode, the following steps are required to define the starting RAM access pointer location:

- Set the page start address of the target display location by command B0h to B7h.
- Set the lower start column address of pointer by command 00h~0Fh.
- Set the upper start column address of pointer by command 10h~1Fh.

For example, if the page address is set to B2h, lower column address is 03h and upper column address is 10h, then that means the starting column is SEG3 of PAGE2. The RAM access pointer is located as shown in Figure 10-2. The input data byte will be written into RAM position of column 3.

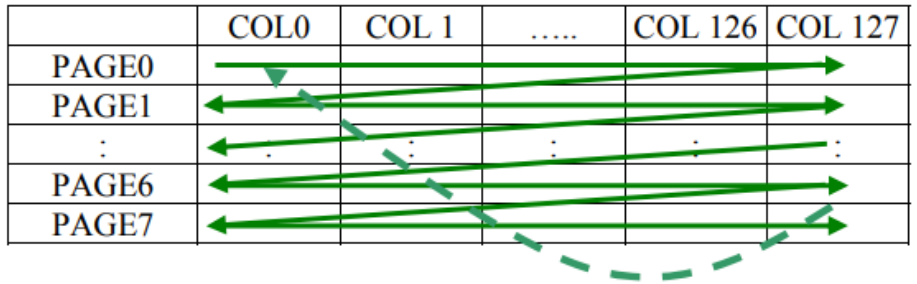
Figure 10-2 : Example of GDDRAM access pointer setting in Page Addressing Mode (No row and column-remapping)



Horizontal addressing mode (A[1:0]=00b)

In horizontal addressing mode, after the display RAM is read/written, the column address pointer is increased automatically by 1. If the column address pointer reaches column end address, the column address pointer is reset to column start address and page address pointer is increased by 1. The sequence of movement of the page and column address point for horizontal addressing mode is shown in Figure 10-3. When both column and page address pointers reach the end address, the pointers are reset to column start address and page start address (Dotted line in Figure 10-3.)

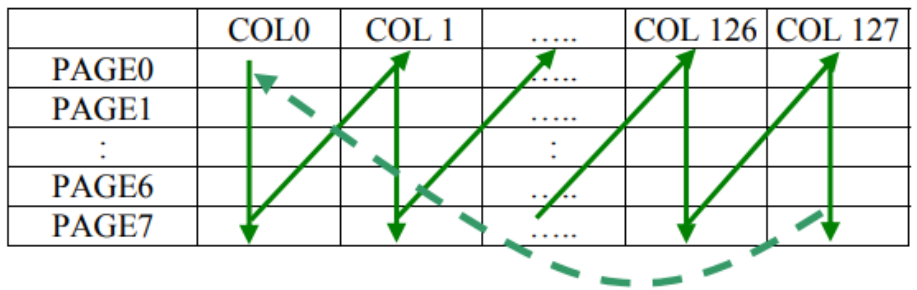
Figure 10-3 : Address Pointer Movement of Horizontal addressing mode



Vertical addressing mode: (A[1:0]=01b)

In vertical addressing mode, after the display RAM is read/written, the page address pointer is increased automatically by 1. If the page address pointer reaches the page end address, the page address pointer is reset to page start address and column address pointer is increased by 1. The sequence of movement of the page and column address point for vertical addressing mode is shown in Figure 10-4. When both column and page address pointers reach the end address, the pointers are reset to column start address and page start address (Dotted line in Figure 10-4.)

Figure 10-4 : Address Pointer Movement of Vertical addressing mode

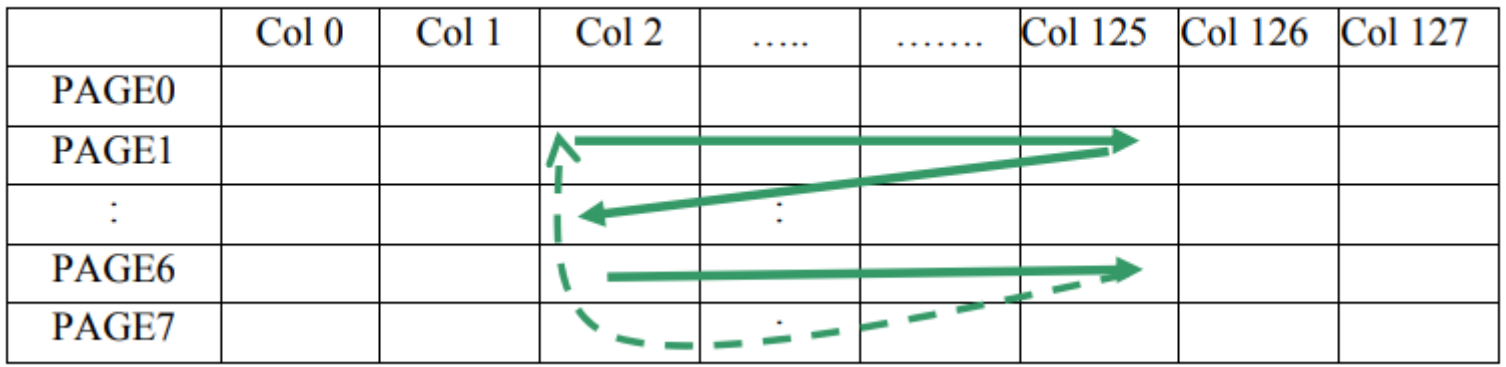


In normal display data RAM read or write and horizontal / vertical addressing mode, the following steps are required to define the RAM access pointer location:

- Set the column start and end address of the target display location by command 21h.
- Set the page start and end address of the target display location by command 22h.

Example is shown in Figure 10-5.

Figure 10-5 : Example of Column and Row Address Pointer Movement



10.2.2 Continuous Vertical and Horizontal Scroll Setup (29h/2Ah)

This command consists of 6 consecutive bytes to set up the continuous vertical scroll parameters and determines the scrolling start page, end page, scrolling speed and vertical scrolling offset.

The bytes B[2:0], C[2:0] and D[2:0] of command 29h/2Ah are for the setting of the continuous horizontal scrolling. The byte E[5:0] is for the setting of the continuous vertical scrolling offset. All these bytes together are for the setting of continuous diagonal (horizontal + vertical) scrolling. If the vertical scrolling offset byte E[5:0] is set to zero, then only horizontal scrolling is performed (like command 26/27h).

Before issuing this command the scroll must be deactivated (2Eh). Otherwise, RAM content may be corrupted. The following figure (Figure 10-10) show the example of using the continuous vertical and horizontal scroll:

Conditions:

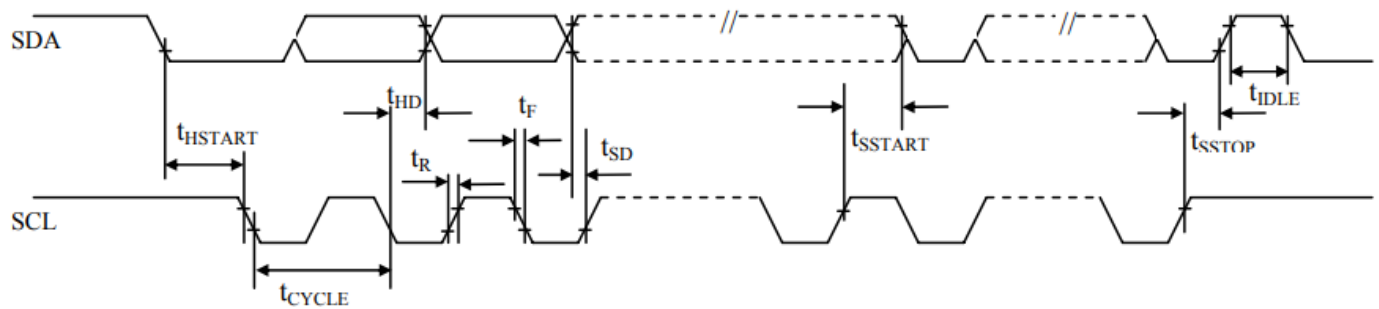
V_{DD} - V_{SS} = V_{DD} - V_{SS} = 1.65V to 3.3V

T_A = 25°C

Table 13-6 :I²C Interface Timing Characteristics

Symbol	Parameter	Min	Typ	Max	Unit
t_{cycle}	Clock Cycle Time	2.5	-	-	us
t_{HSTART}	Start condition Hold Time	0.6	-	-	us
t_{HD}	Data Hold Time (for "SDA_{OUT}" pin)	0	-	-	ns
	Data Hold Time (for "SDA_{IN}" pin)	300	-	-	ns
t_{SD}	Data Setup Time	100	-	-	ns
t_{SSTART}	Start condition Setup Time (Only relevant for a repeated Start condition)	0.6	-	-	us
t_{SSTOP}	Stop condition Setup Time	0.6	-	-	us
t_R	Rise Time for data and clock pin	-	-	300	ns
t_F	Fall Time for data and clock pin	-	-	300	ns
t_{IDLE}	Idle Time before a new transmission can start	1.3	-	-	us

Figure 13-5 : I²C interface Timing characteristics



```python
import time
import board
import busio
import adafruit_ssd1306
# https://github.com/adafruit/Adafruit_CircuitPython_Bundle/releases/download/20240910/adafruit-circuitpython-bundle-9.x-mpy-20240910.zip
# https://github.com/adafruit/Adafruit_CircuitPython_Bundle/releases/
# Extract the files 'adafruit_ssd1306.mpy', 'adafruit_framebuf.mpy' and save in the dir '/lib'
SCL, SDA = board.GP17, board.GP16
i2c = busio.I2C(SCL, SDA)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
display.fill(0)
display.show()
draw = [
    "          **          ",
    "       **    **       ",
    "    **          **     ",
    "  **              **   ",
    " **                ** ",
    " **    **  **      ** ",
    " **                ** ",
    "  **              **  ",
    "    **          **    ",
    "       **    **       ",
    "          **          ",
    ]

rows = 32#len(draw)
columns = len(draw[0])
columns2=128
print('rows,columns',rows,columns)
image_bytearray = bytearray(rows * columns)
image_bytearray[0]=columns
for _ in range(100):
    for l in range(10):
        for i in range(0,rows,8):
            for j in range(columns2):
                #print('i,j,k',i,j)
                a=sum(0 if draw[i+k+l][j%columns]==" " else 2**k for k in range(8))
                display.buffer[i*columns2//8+j+1]=a
        display.show()
        #time.sleep(.05)
```

```
# SPDX-FileCopyrightText: 2017 Michael McWethy for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
`adafruit_ssd1306`
====================================================

MicroPython SSD1306 OLED driver, I2C and SPI interfaces

* Author(s): Tony DiCola, Michael McWethy
"""

import time

from micropython import const
from adafruit_bus_device import i2c_device, spi_device

try:
except ImportError:
    # CircuitPython framebuf import
    import adafruit_framebuf as framebuf

    _FRAMEBUF_FORMAT = framebuf.MVLSB

try:
    # Used only for typing
    from typing import Optional
    import busio
    import digitalio
except ImportError:

__version__ = "2.12.17"
__repo__ = "https://github.com/adafruit/Adafruit_CircuitPython_SSD1306.git"
```

```
# register definitions
SET_CONTRAST = const(0x81)
SET_ENTIRE_ON = const(0xA4)
SET_NORM_INV = const(0xA6)
SET_DISP = const(0xAE)
SET_MEM_ADDR = const(0x20)
SET_COL_ADDR = const(0x21)
SET_PAGE_ADDR = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP = const(0xA0)
SET_MUX_RATIO = const(0xA8)
SET_IREF_SELECT = const(0xAD)
SET_COM_OUT_DIR = const(0xC0)
SET_DISP_OFFSET = const(0xD3)
SET_COM_PIN_CFG = const(0xDA)
SET_DISP_CLK_DIV = const(0xD5)
SET_PRECHARGE = const(0xD9)
SET_VCOM_DESEL = const(0xDB)
SET_CHARGE_PUMP = const(0x8D)
```

```
222 class SSD1306_I2C(_SSD1306):
223     """
224     I2C class for SSD1306
225 
226     :param width: the width of the physical screen in pixels,
227     :param height: the height of the physical screen in pixels,
228     :param i2c: the I2C peripheral to use,
229     :param addr: the 8-bit bus address of the device,
230     :param external_vcc: whether external high-voltage source is connected.
231     :param reset: if needed, DigitalInOut designating reset pin
232     """
233 
234     def   init  (
244     ):
264 
265     def write_cmd(self, cmd: int) -> None:
271 
272     def write_framebuf(self) -> None:
```

```
 62  class _SSD1306(framebuf.FrameBuffer):
 63      """Base class for SSD1306 display driver"""
 64
 65      # pylint: disable-msg=too-many-arguments
 66      def   init  (
 75      ):
104
105      @property
106      def power(self) -> bool:
109
110      def init display(self) -> None:
161
162      def poweroff(self) -> None:
166
167      def contrast(self, contrast: int) -> None:
171
172      def invert(self, invert: bool) -> None:
175
176      def rotate(self, rotate: bool) -> None:
180          # com output (vertical mirror) is changed immediately
181          # you need to call show() for the seg remap to be visible
182
183      def write framebuf(self) -> None:
186
187      def write cmd(self, cmd: int) -> None:
190
191      def poweron(self) -> None:
202
203      def show(self) -> None:
```

```
278 class FrameBuffer:
279     """FrameBuffer object.
280
281     :param buf: An object with a buffer protocol which must be large enough to contain every
282                 pixel defined by the width, height and format of the FrameBuffer.
283     :param width: The width of the FrameBuffer in pixel
284     :param height: The height of the FrameBuffer in pixel
285     :param buf_format: Specifies the type of pixel used in the FrameBuffer; permissible values
286                        are listed under Constants below. These set the number of bits used to
287                        encode a color value and the layout of these bits in ``buf``. Where a
288                        color value c is passed to a method, c is  a small integer with an encoding
289                        that is dependent on the format of the FrameBuffer.
290     :param stride: The number of pixels between each horizontal line of pixels in the
291                    FrameBuffer. This defaults to ``width`` but may need adjustments when
292                    implementing a FrameBuffer within another larger FrameBuffer or screen. The
293                    ``buf`` size must accommodate an increased step size.
294
295     """
296
297     def   init  (self, buf, width, height, buf format=MVLSB, stride=None):
319
320     @property
321     def rotation(self):
324
325     @rotation.setter
326     def rotation(self, val):
```

```
331     def fill(self, color):
334
335     def fill_rect(self, x, y, width, height, color):
340
341     def pixel(self, x, y, color=None):
360
361     def hline(self, x, y, width, color):
364
365     def vline(self, x, y, height, color):
368
369     def circle(self, center_x, center_y, radius, color):
394
395     def rect(self, x, y, width, height, color, *, fill=False):
432
433     def line(self, x_0, y_0, x_1, y_1, color):
460
461     def blit(self):
464
465     def scroll(self, delta_x, delta_y):
491
492     # pylint: disable=too-many-arguments
493     def text(self, string, x, y, color, *, font_name="font5x8.bin", size=1):
520
521     # pylint: enable=too-many-arguments
522
523     def image(self, img):
```