

PROGRAMMING THE CHARACTER GENERATOR RAM (CG RAM)

The character generator RAM (CG RAM) allows the user to create up to eight custom 5 x 7 characters + cursor (5 x 8). Once programmed, the custom characters or symbols are accessed exactly as if they were in ROM. However since the RAM is a volatile memory, power must be continually maintained. Otherwise, the custom characters/symbols must be programmed into non-volatile external ROM and sent to the display after each display initialization. All dots in the 5 x 8 dot matrix can be programmed, which includes the cursor position.

The modules RAM are divided into two parts: data display RAM (DD RAM) and custom character generator RAM (CG RAM). This is not to be confused programming the custom character generator RAM with the 192 character generator ROM. The CG RAM is located between hex 40 and 7F and is contiguous. Locations 40 thru 47 hold the first custom character (5 x 8), 48 thru 4F hold the second custom character, 50 thru 57 hold the third CG, and so forth to 78 thru 7F for the eighth CG character/symbol.

If during initialization the display was programmed to automatically increment, then only the single initial address, 40, need be sent. Consecutive row data will automatically appear at 41, 42, etc. until the completed character is formed. All eight custom CG characters can be programmed in 64 consecutive "writes" after sending the single initial 40 address.

The CG RAM is 8 bits wide, although only the right-most 5-bits are used for a custom CG character row. The left-most dot of programming the CG RAM character corresponds to D4 in the most significant nibble (XXXD4) of the data bus code, with the remaining 4 dots in the row corresponding to the least significant nibble (D3 thru D0), D0 being the right-most dot. Thus, hex 1F equals all dots on and hex 00 equals all dots off. Examples include hex 15 (10101) equal to 3 dots on the hex 0A (01010) equal 2 dots on. In each case the key 5-bits of the 8-bit code program one row of a custom CG character. When all 7 or 8 rows are programmed, the character is complete. A graphic example is shown below:

| RS | R/W | Data | Display | Description |
|----|-----|------|---------|---|
| 0 | 0 | 40 | — | addresses 1st row, 1st CG character |
| 1 | 0 | 11 | * * | result of 11, 1st row |
| 1 | 0 | 0A | ** | result of 0A, 2nd row |
| 1 | 0 | 1F | ***** | result of 1F, 3rd row |
| 1 | 0 | 04 | * | result of 04, 4th row |
| 1 | 0 | 1F | ***** | result of 1F, 5th row |
| 1 | 0 | 04 | * | result of 04, 6th row |
| 1 | 0 | 04 | * | result of 04, 7th row |
| 1 | 0 | 00 | — | result of 00, 8th row (cursor position) |
| 1 | 0 | 15 | *** | 1st row, 2nd CG character. Note: Addressing not now required; hex 48 is next in the sequence. |

5 x 7 + CURSOR

Relationships between CG RAM addresses and character codes (DD RAM) and character patterns (CG RAM data),
(5 x 7 dot matrix).5 X 7 Table

| Character code (DD RAM data) | | | | | | | | CG RAM address | | | | Character pattern (CG RAM data) | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------|---|---|---|-----------|---|---|---|----------------|---|---|---|---------------------------------|---|---|---|-----------|---|---|---|-----------|---|---|--|--|--|--|--|--|--|--|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| Upper bit | | | | Lower bit | | | | Upper bit | | | | Lower bit | | | | Upper bit | | | | Lower bit | | | | | | | | | | | | | | | |
| 0 0 0 0 * 0 0 0 | | | | | | | | 0 0 0 | | | | 0 | 0 | 0 | * | * | * | 1 | 1 | 1 | 1 | 0 | <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> * * * ↑ ↓ * * * </div> <div style="border: 1px solid black; padding: 2px;"> 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 </div> </div> | | | | | | | | | | | | |
| | | | | | | | | | | | | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | | | | | | | | | 1 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | | | | | | | | | 1 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | | | | | | | | | 0 | 1 | 0 | 0 | 0 |
| | | | | | | | | | | | | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | | | | | | | | | 0 | 0 | 0 | 1 | 0 |
| | | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | 0 | 0 | 1 | 0 | 0 |
| | | | | | | | | | | | | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | 0 | 0 | 0 | 1 | 0 |
| | | | | | | | | | | | | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | | | | | | | | | 0 | 0 | 0 | 0 | 1 |
| 0 0 0 0 * 0 0 1 | | | | | | | | 0 0 1 | | | | 0 | 0 | 0 | * | * | * | 1 | 0 | 0 | 0 | 1 | <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> * * * ↑ ↓ * * * </div> <div style="border: 1px solid black; padding: 2px;"> 0 1 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 </div> </div> | | | | | | | | | | | | |
| | | | | | | | | | | | | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | | | | | | | | | 0 | 1 | 0 | 0 | |
| | | | | | | | | | | | | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | | | | | | | | 0 | 1 | 0 | 0 | |
| | | | | | | | | | | | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | | | | | | | | | 0 | 1 | 0 | 0 | |
| | | | | | | | | | | | | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | | | | | | | | | 0 | 0 | 0 | 0 | |
| | | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | 0 | 0 | 1 | 0 | |
| | | | | | | | | | | | | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | 0 | 0 | 0 | 0 | |
| | | | | | | | | | | | | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | | | | | | | | | 0 | 0 | 0 | 0 | |
| 0 0 0 0 * 1 1 1 | | | | | | | | 1 1 1 | | | | 0 | 0 | 0 | * | * | * | 1 | 1 | 1 | 0 | 0 | <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> * * * ↑ ↓ * * * </div> <div style="border: 1px solid black; padding: 2px;"> 1 0 0 1 0 1 1 1 0 1 1 1 </div> </div> | | | | | | | | | | | | |
| | | | | | | | | | | | | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | 0 | 0 | 0 | | |
| | | | | | | | | | | | | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | | | | | | | | 0 | 0 | 0 | | |
| | | | | | | | | | | | | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | | | | | | | | | 0 | 0 | 0 | | |

Example of character pattern (R)

Cursor position

Example of character pattern (¥)

- NOTES:**
- ▶ In CG RAM data, 1 corresponds to Selection and 0 to Non-selection on the display.
 - ▶ Character code bits 0 to 2 and CG RAM address bits 3 to 5 correspond with each other (three bits, eight types).
 - ▶ CG RAM address bits 0 to 2 specify a line position for a character pattern. Line 8 of a character pattern is the cursor position where the logical sum of the cursor and CG RAM data is displayed. Set the data of line 8 to 0 to display the cursor. If the data is charged to 1, one bit lights, regardless of the cursor.
 - ▶ The character pattern column position corresponds to CG RAM data bits 0 to 4 and bit 4 comes to the left end. CG RAM data bits 5 to 7 are not displayed but can be used as general data RAM.
 - ▶ When reading a character pattern from CG RAM, set to 0 all of character code bits 4 to 7. Bits 0 to 2 determine which pattern will be read out. Since bit 3 is not valid, 00H and 08H select the same character.