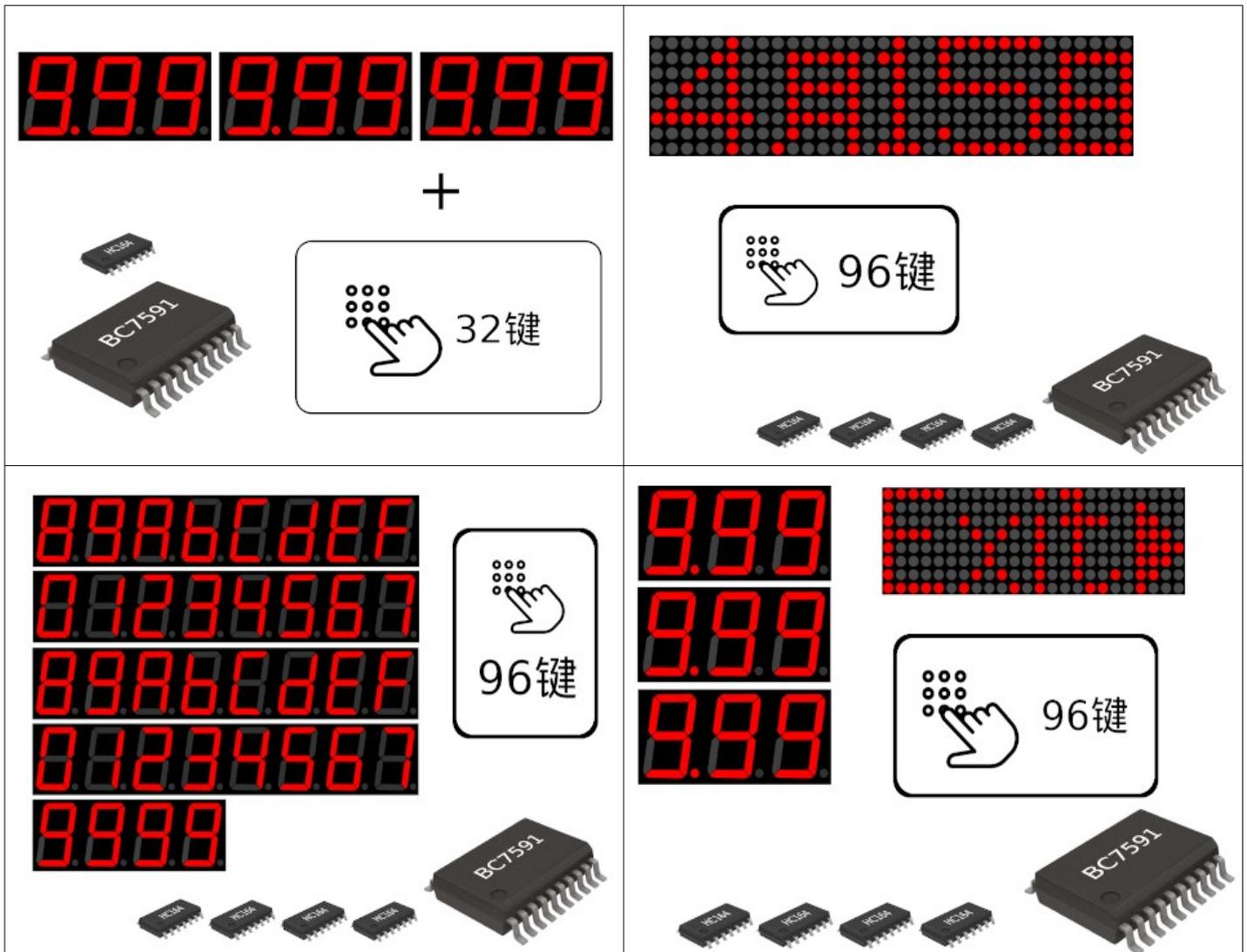


# BC7591

## 256 Segments LED Driver With 96-key Keyboard Interface

- Configurable 64-256 Segments Display
- 32-96 Keys Keyboard
- Built-in 7-segment Decoder
- 128 Individually Blinkable LEDs, Adjustable Blink Frequency
- Shift Left/Shift Right Commands
- 16-Level Dim Control
- All LEDs Individually Addressable
- Supports Combined Key And Long-press
- Direct Key Value Output
- UART Interface
- 3.0-5.5V
- SSOP20 Package
- Protocol Compatible With BC6xxx
- Works With Both Normally Open And Normally Closed Switches

### Typical Applications:



## Overview

BC7591 is a LED display driver with keyboard matrix interface, configurable by the number of 74HC164 shift-registers attached, it can drive from 64 to 256 segments of LED display while providing 32-96 keys keyboard interface. All the LEDs are individually addressable, the first 128 LEDs can self-blink at user configurable frequency.

A built-in 7-segment decoder can be used for driving 7 segments displays, the decimal points are separately controlled and can be used for extra digits, special commands can be used to distribute the output of the 7-segment decoder to the DP segments of each digit. With 1 8-bit shift register, a maximum of 9 digits with minus sign can be driven, and so on with 2 shift registers 18 digits and 4 for as many as 36 digits.

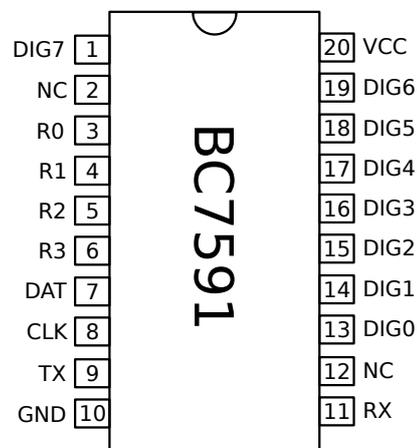
BC7591 can also be used for dot matrix display, or a combination of 7-segment display and dot matrices. When used as dot matrix driver, the maximum resolution is 32x8. BC7591 has dedicated commands for dot matrix displays such as left/right and up/down shift move of the display frame.

Based on the number of shift registers used, keyboard can vary from 32 keys to 96 keys. Both normally open and normally closed switches can be used in circuits, and both combined keys and long-press keys are supported.

BC7591 uses 1/8 duty cycle scanning, comparing with 1/16, the switching frequency and peak current are reduced by 50% therefore generates less noise. BC7591 has 16 dim levels.

Using UART as communication interface, it can be easily connected with a wide range of micro-controllers or be controlled by computers directly (via a USB-UART converter). It is also easy to add isolation or to connect a RS-485/RS-422 network to create remote consoles with UART.

## Pin Configuration



## Pin Descriptions

VCC	20	Power supply, 3.0~5.5V
GND	10	Ground
TX	9	UART Tx, Open Drain
RX	11	UART Rx
R0~R3	3,4,5,6	Key matrix rows, with internal pull-ups
DIG0-DIG7	1,13,14,15,16,17,18,19	Digit or dot matrix row drives, active low, high-z when inactive

DAT	7	shift register data
CLK	8	shift register clock
NC	2,12	no connection

(Table 1: Pin Descriptions)

## Display Driver

### Display Types

BC7591 can drive 7-segment displays, or LED dot matrices, or a combination of these two. From the BC7591's point of view, there will be no difference between different LED display components, but from the user's point of view, if you accidentally used a command which is designed to be used with a different display type, the display will be look like corrupted and meaningless, such as if you use a 7-segment decode command on a dot matrix display.

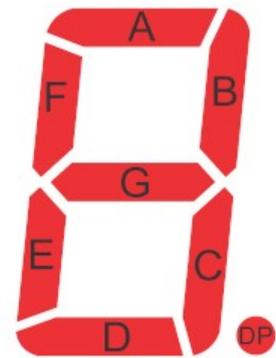
### 7-segment Display

Definitions of segments on a 7-segment display:

There are 2 ways to connect a 7-segment display: 7-segment without decimal point or 8-segment with decimal point. In many scenarios, decimal points are at fixed position or not needed at all, so these segments for decimal points are free to be used as independent indicators or to drive an extra digit.

A 7-segment decoder is embedded in BC7591, which can translate a value from 0 to 15 to 7-segment display of '0' to 'F'. Please refer to "DECODE\_WT" command for more information. There is also a dedicated decode command for the extra digits driven by the spare decimal points. All the decode commands are 7 segments decoding, which means when you apply a decode command on a digit, only segments A to G are updated, leave the DPs unchanged.

Digit lines, sometimes called common lines, are driven by BC7591 directly, while the segment lines, or the column lines in a dot matrix display, are driven by the output of the shift registers. Q0 of the shift register is for segment A, and Q7 will be connected to segment DP.



The sink current at DIG lines on BC7591 is 100mA, so when the segment is not more than 64 (one 8-bit shift register), the digit lines can be driven by BC7591 directly without any external components. If more segments are needed, external PNP type transistors should be used as line drivers.

The brightness of the LED is determined by the efficiency of LED and current. To keep a clear and distinguishable display, the LED should have a 3mcd of intensity at 10mA at least.

The current through LEDs is decided by the driver capability and current limiting resistors. The resistor value can be calculated by

$$R = (V_{\text{seg}} - V_{\text{LED}} - V_{\text{DIG}}) / I_{\text{seg}}$$

Where  $V_{\text{seg}}$  is the voltage at shift register Qn when the its output current is the LED's forward current  $I_{\text{seg}}$ , and  $V_{\text{LED}}$  is the LED's forward voltage,  $V_{\text{DIG}}$  is the voltage on digit driver at current  $8 \cdot I_{\text{seg}}$ .

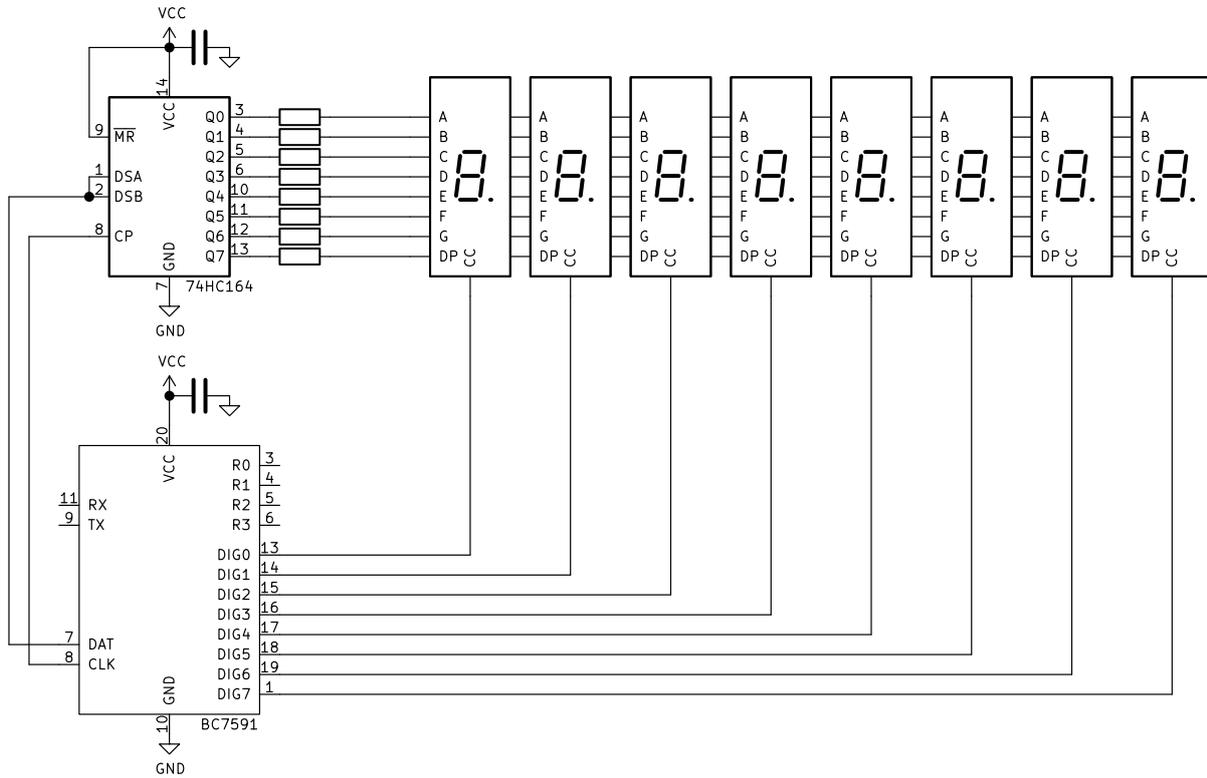
Example1: When power supply voltage is 5V, with the condition  $I_{\text{seg}}=12\text{mA}$ ,  $V_{\text{LED}}=2\text{V}$ ,  $V_{\text{seg}}=4.2\text{V}(@12\text{mA})$  and  $V_{\text{DIG}}=1\text{V}(@96\text{mA})$ , the current limiting resistors should be  $R=(4.2-2.0-1)/0.012 = 100\Omega$ .

Example2: Power supply voltage is 3.3V, with the condition  $I_{\text{seg}}=10\text{mA}$ ,  $V_{\text{LED}}=1.9\text{V}$ ,  $V_{\text{SEG}}=2.8\text{V}(@10\text{mA})$  and  $V_{\text{DIG}}=0.7\text{V}(@80\text{mA})$ , the current limiting resistors should be  $R=(2.8-1.9-0.7)/0.01 = 20\Omega$ .

### Application: 8 digits with 8-segment connection

BC7591's internal driver is capable to drive 8 7-segment digits or 64 LEDs, therefore when the total segment count is less than 65 the circuit is in its simplest form, only one 74HC164 shift register is needed. In the following circuit, every

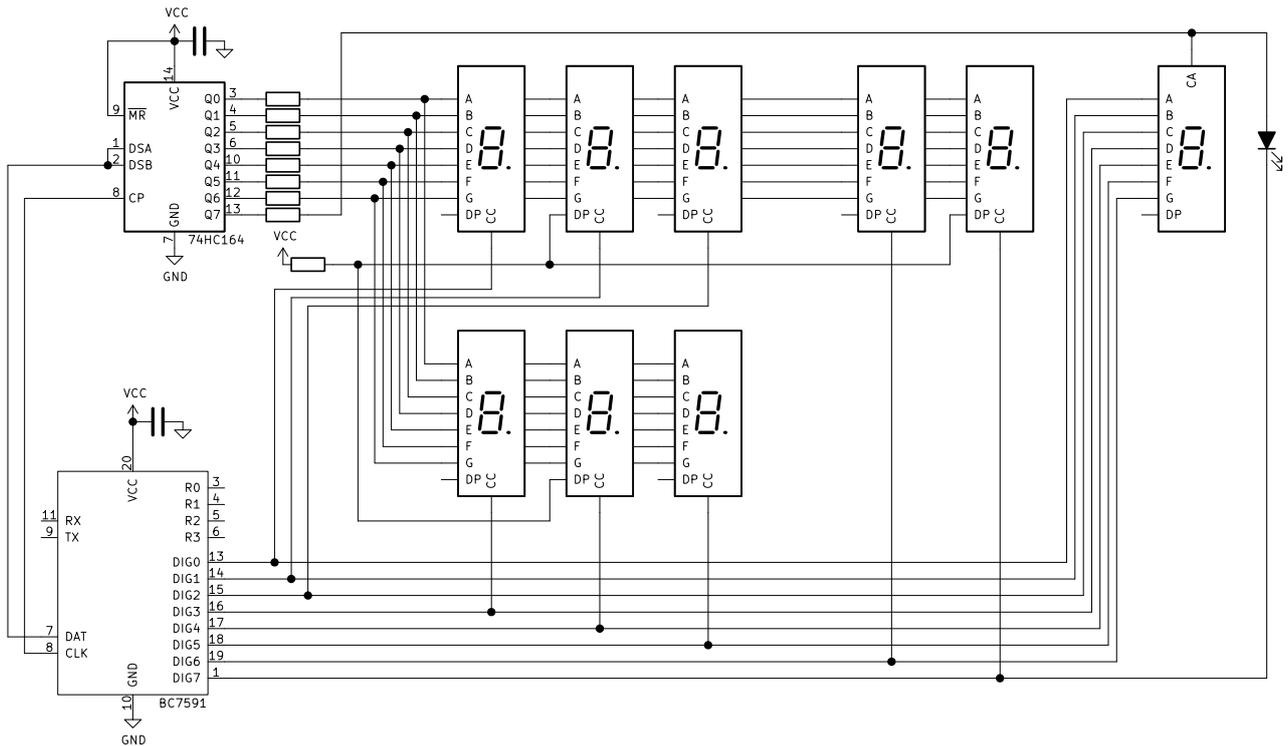
digit has its own controllable decimal point. (No keyboard circuit is shown in the diagram, please refer to the keyboard section later in the article)



(Diagram1: BC7591-8 Digits with 8-segment connection)

### **Application: 9 digits with 7-segment connection**

When decimal points are at fixed position or not used at all, DP segments on each digit can be used to drive extra displays. Note, in the following circuit, the extra digit is common-anode, different from the other common-cathode modules. In this diagram, 3 decimal points are at fixed position, so the Q7 output is used to drive an extra digit. Comparing to Diagram1, this circuit can drive 1 more digit of display and an independent LED indicator with same amount of components.

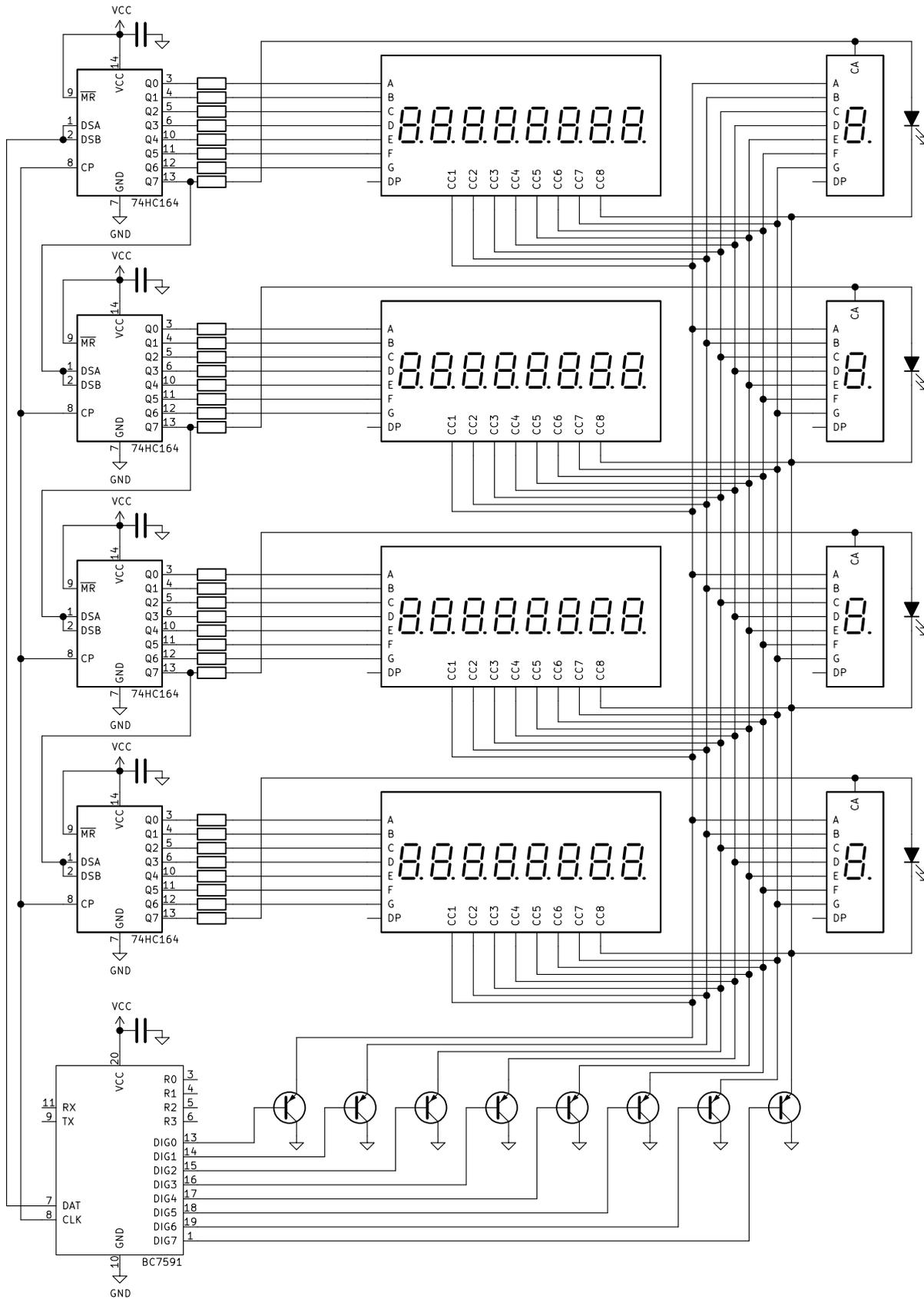


(Diagram2: BC7591-9 digits with 7-segment connection)

### **Application: 36 digits with 7-segment connection**

When the segments to be driven are more than 64, the internal driver will not have sufficient current if used directly, an external driver must be used, usually it is as simple as a NPN transistor, no resistor is needed.

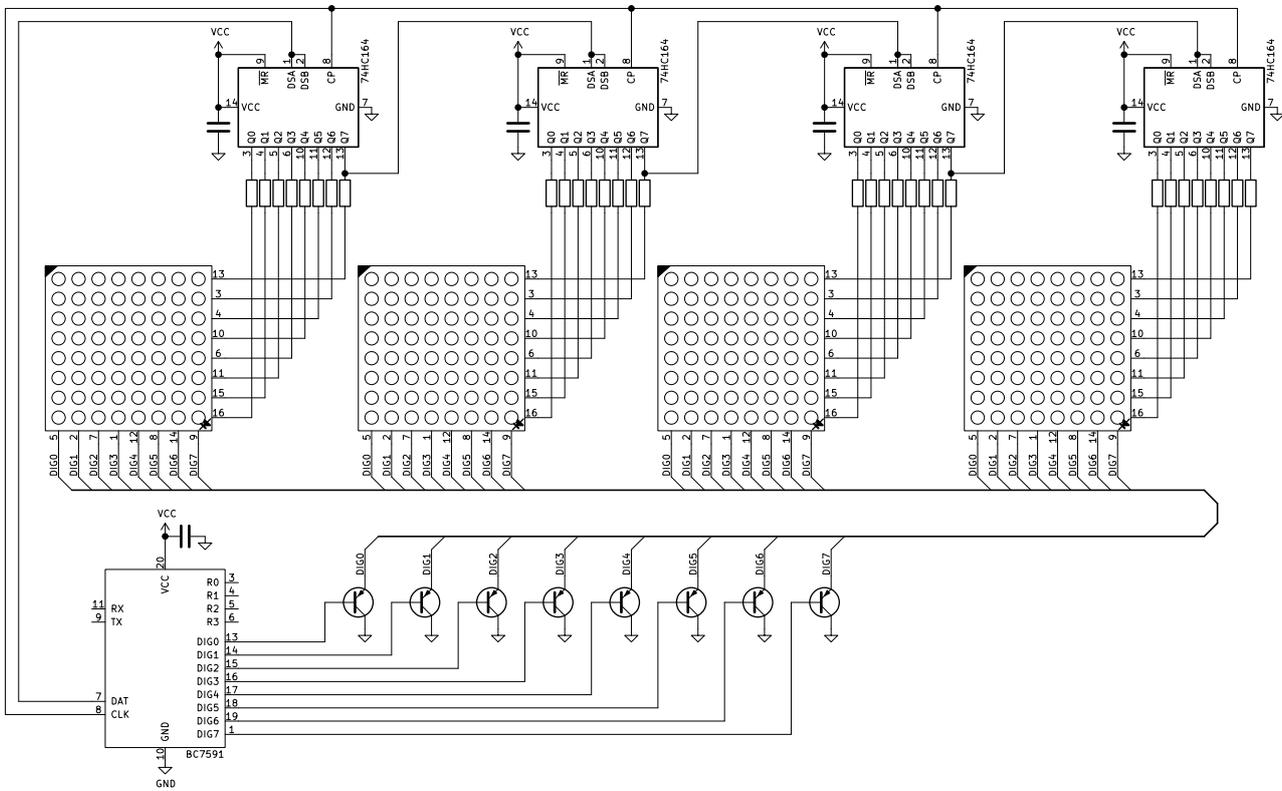
The following diagram shows a BC7591 drives 36 7-segment displays.



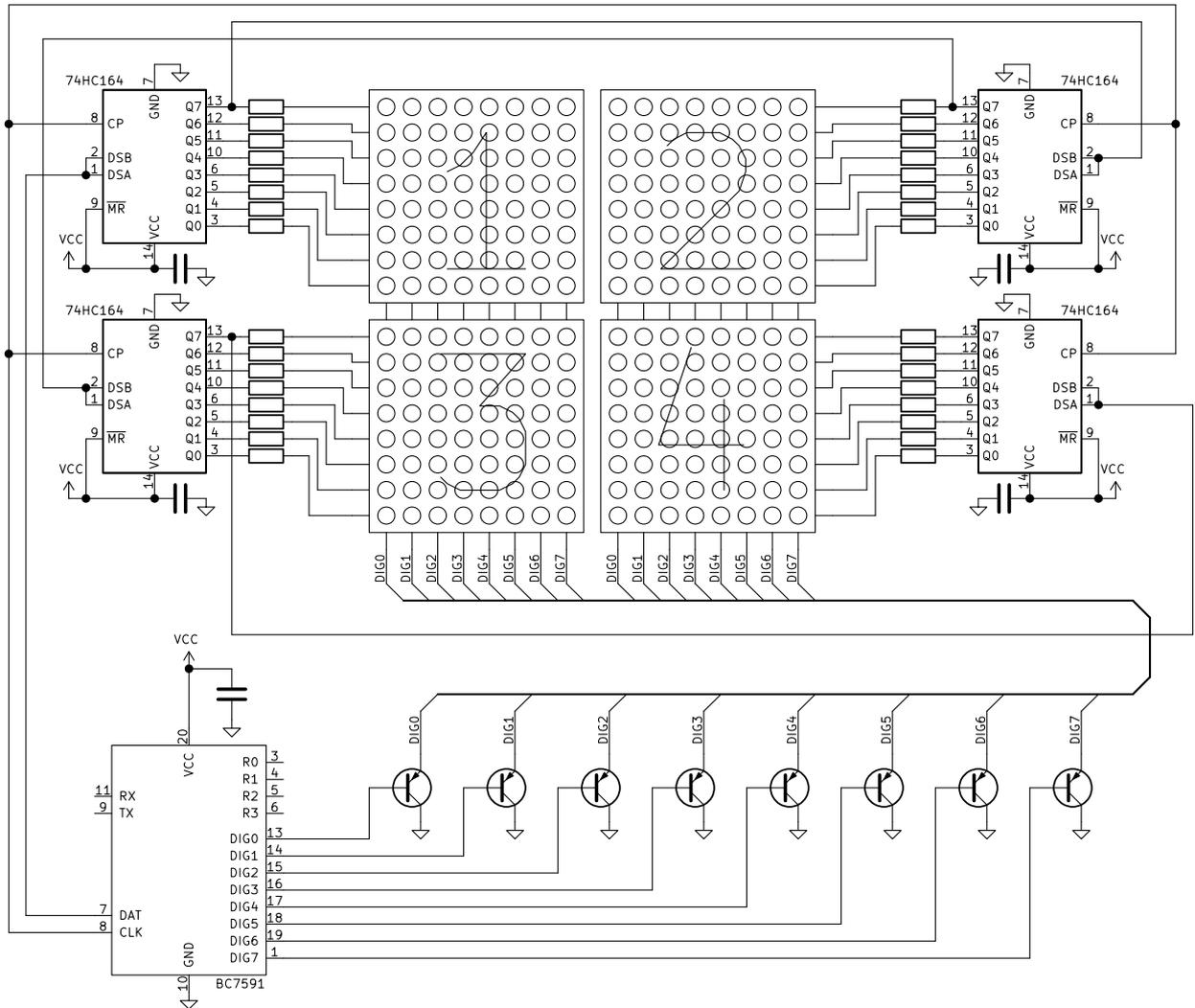
(Diagram 3: BC7591-36 digits 7-segment connection)

## Application: Dot Matrix Display

When used as a dot matrix display driver, the shift register outputs will drive the matrix columns, and the digit lines will be the rows. The maximum dimension of the matrix will be 32\*8. The matrix can be arranged as 32\*8 or 16\*16 as per requirements, there will be no difference between the two configurations in the circuit aspect.



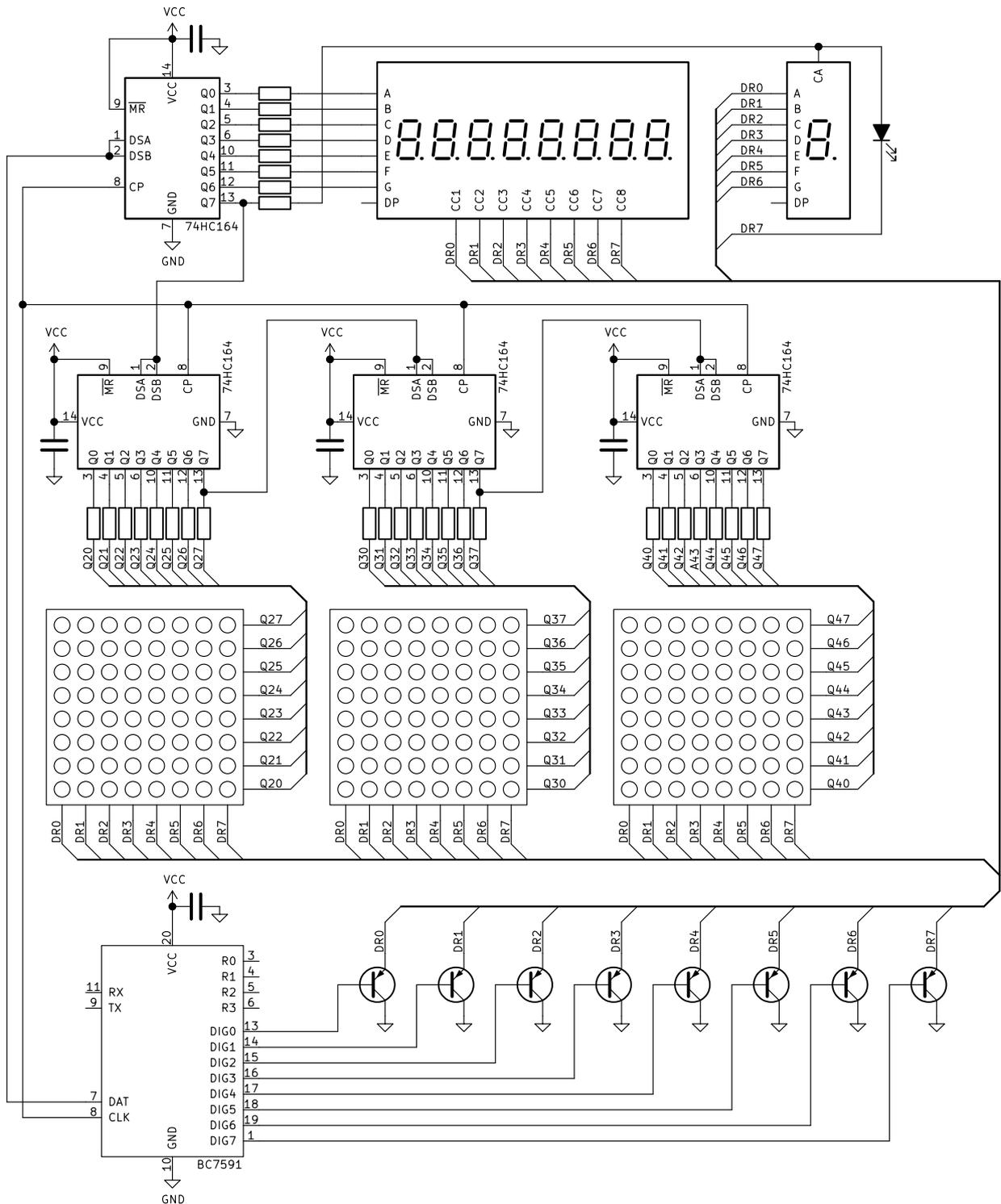
(Diagram 4: BC7591-32x8 matrix)



(Diagram 5: BC7591-16x16 matrix)

## Application: Mixed Display

The following diagram shows a circuit with 9 digits 7-segment display with 24\*8 dot matrix, driven by a single BC7591.



(Diagram 6: BC7591-9 digits 7-segment display+24\*8 matrix)



Display Registers																																
	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
b <sub>7</sub>	A <sub>x1</sub>	B <sub>x1</sub>	C <sub>x1</sub>	D <sub>x1</sub>	E <sub>x1</sub>	F <sub>x1</sub>	G <sub>x1</sub>	L <sub>1</sub>	A <sub>x2</sub>	B <sub>x2</sub>	C <sub>x2</sub>	D <sub>x2</sub>	E <sub>x2</sub>	F <sub>x2</sub>	G <sub>x2</sub>	L <sub>2</sub>	A <sub>x3</sub>	B <sub>x3</sub>	C <sub>x3</sub>	D <sub>x3</sub>	E <sub>x3</sub>	F <sub>x3</sub>	G <sub>x3</sub>	L <sub>3</sub>	A <sub>x4</sub>	B <sub>x4</sub>	C <sub>x4</sub>	D <sub>x4</sub>	E <sub>x4</sub>	F <sub>x4</sub>	G <sub>x4</sub>	L <sub>4</sub>
b <sub>6</sub>	G <sub>1</sub>	G <sub>2</sub>	G <sub>3</sub>	G <sub>4</sub>	G <sub>5</sub>	G <sub>6</sub>	G <sub>7</sub>	G <sub>8</sub>	G <sub>9</sub>	G <sub>10</sub>	G <sub>11</sub>	G <sub>12</sub>	G <sub>13</sub>	G <sub>14</sub>	G <sub>15</sub>	G <sub>16</sub>	G <sub>17</sub>	G <sub>18</sub>	G <sub>19</sub>	G <sub>20</sub>	G <sub>21</sub>	G <sub>22</sub>	G <sub>23</sub>	G <sub>24</sub>	G <sub>25</sub>	G <sub>26</sub>	G <sub>27</sub>	G <sub>28</sub>	G <sub>29</sub>	G <sub>30</sub>	G <sub>31</sub>	G <sub>32</sub>
b <sub>5</sub>	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	F <sub>4</sub>	F <sub>5</sub>	F <sub>6</sub>	F <sub>7</sub>	F <sub>8</sub>	F <sub>9</sub>	F <sub>10</sub>	F <sub>11</sub>	F <sub>12</sub>	F <sub>13</sub>	F <sub>14</sub>	F <sub>15</sub>	F <sub>16</sub>	F <sub>17</sub>	F <sub>18</sub>	F <sub>19</sub>	F <sub>20</sub>	F <sub>21</sub>	F <sub>22</sub>	F <sub>23</sub>	F <sub>24</sub>	F <sub>25</sub>	F <sub>26</sub>	F <sub>27</sub>	F <sub>28</sub>	F <sub>29</sub>	F <sub>30</sub>	F <sub>31</sub>	F <sub>32</sub>
b <sub>4</sub>	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	E <sub>4</sub>	E <sub>5</sub>	E <sub>6</sub>	E <sub>7</sub>	E <sub>8</sub>	E <sub>9</sub>	E <sub>10</sub>	E <sub>11</sub>	E <sub>12</sub>	E <sub>13</sub>	E <sub>14</sub>	E <sub>15</sub>	E <sub>16</sub>	E <sub>17</sub>	E <sub>18</sub>	E <sub>19</sub>	E <sub>20</sub>	E <sub>21</sub>	E <sub>22</sub>	E <sub>23</sub>	E <sub>24</sub>	E <sub>25</sub>	E <sub>26</sub>	E <sub>27</sub>	E <sub>28</sub>	E <sub>29</sub>	E <sub>30</sub>	E <sub>31</sub>	E <sub>32</sub>
b <sub>3</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	D <sub>8</sub>	D <sub>9</sub>	D <sub>10</sub>	D <sub>11</sub>	D <sub>12</sub>	D <sub>13</sub>	D <sub>14</sub>	D <sub>15</sub>	D <sub>16</sub>	D <sub>17</sub>	D <sub>18</sub>	D <sub>19</sub>	D <sub>20</sub>	D <sub>21</sub>	D <sub>22</sub>	D <sub>23</sub>	D <sub>24</sub>	D <sub>25</sub>	D <sub>26</sub>	D <sub>27</sub>	D <sub>28</sub>	D <sub>29</sub>	D <sub>30</sub>	D <sub>31</sub>	D <sub>32</sub>
b <sub>2</sub>	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>	C <sub>8</sub>	C <sub>9</sub>	C <sub>10</sub>	C <sub>11</sub>	C <sub>12</sub>	C <sub>13</sub>	C <sub>14</sub>	C <sub>15</sub>	C <sub>16</sub>	C <sub>17</sub>	C <sub>18</sub>	C <sub>19</sub>	C <sub>20</sub>	C <sub>21</sub>	C <sub>22</sub>	C <sub>23</sub>	C <sub>24</sub>	C <sub>25</sub>	C <sub>26</sub>	C <sub>27</sub>	C <sub>28</sub>	C <sub>29</sub>	C <sub>30</sub>	C <sub>31</sub>	C <sub>32</sub>
b <sub>1</sub>	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>5</sub>	B <sub>6</sub>	B <sub>7</sub>	B <sub>8</sub>	B <sub>9</sub>	B <sub>10</sub>	B <sub>11</sub>	B <sub>12</sub>	B <sub>13</sub>	B <sub>14</sub>	B <sub>15</sub>	B <sub>16</sub>	B <sub>17</sub>	B <sub>18</sub>	B <sub>19</sub>	B <sub>20</sub>	B <sub>21</sub>	B <sub>22</sub>	B <sub>23</sub>	B <sub>24</sub>	B <sub>25</sub>	B <sub>26</sub>	B <sub>27</sub>	B <sub>28</sub>	B <sub>29</sub>	B <sub>30</sub>	B <sub>31</sub>	B <sub>32</sub>
b <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>	A <sub>6</sub>	A <sub>7</sub>	A <sub>8</sub>	A <sub>9</sub>	A <sub>10</sub>	A <sub>11</sub>	A <sub>12</sub>	A <sub>13</sub>	A <sub>14</sub>	A <sub>15</sub>	A <sub>16</sub>	A <sub>17</sub>	A <sub>18</sub>	A <sub>19</sub>	A <sub>20</sub>	A <sub>21</sub>	A <sub>22</sub>	A <sub>23</sub>	A <sub>24</sub>	A <sub>25</sub>	A <sub>26</sub>	A <sub>27</sub>	A <sub>28</sub>	A <sub>29</sub>	A <sub>30</sub>	A <sub>31</sub>	A <sub>32</sub>

(Table 4: Display Register Bits Map)

### Direct Register Write Command: DIRECT\_WT (0x00 - 0x1F)

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	0	0	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

A<sub>4</sub>:A<sub>0</sub> is the register address, and the second byte D<sub>7</sub>:D<sub>0</sub> is the data to be written in the register. When 7-segment display is used, this command can be used to display special characters such as 'H', or 'L', when driving dot matrix display, this command is used to update the content of a column, A<sub>4</sub>:A<sub>0</sub> is the column address.

This command is the alias of COL\_WRITE command.

### Segment Access Command: SEG\_OFF, SEG\_ON (0xC0, 0xC1)

Segment Off (clear) command: 0xC0:

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	0	0	0	0	0	0	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>

Segment On (set) command: 0xC1:

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	0	0	0	0	0	1	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>

Every bit in BC7591's display register has an individual address, the address is in a continuous space ranged from 0x00 to 0xFF. Using SEG\_OFF and SEG\_ON commands, user can have individual control on each display segment.

Display Registers									
	0x00	0x01	0x02	0x03	...	0x1C	0x1D	0x1E	0x1F
b <sub>7</sub>	0x07	0x0F	0x17	0x1F	...	0xE7	0xEF	0xF7	0xFF
b <sub>6</sub>	0x06	0x0E	0x16	0x1E		0xE6	0xEE	0xF6	0xFE
b <sub>5</sub>	0x05	0x0D	0x15	0x1D		0xE5	0xED	0xF5	0xFD
b <sub>4</sub>	0x04	0x0C	0x14	0x1C		0xE4	0xEC	0xF4	0xFC
b <sub>3</sub>	0x03	0x0B	0x13	0x1B		0xE3	0xEB	0xF3	0xFB
b <sub>2</sub>	0x02	0x0A	0x12	0x1A		0xE2	0xEA	0xF2	0xFA
b <sub>1</sub>	0x01	0x09	0x10	0x19		0xE1	0xE9	0xF0	0xF9
b <sub>0</sub>	0x00	0x08	0x10	0x18		0xE0	0xE8	0xF0	0xF8

(Table 5: Display Register Bit Addresses)

### Write To All Registers Command: *WRITE\_ALL* (0xF1)

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	1	1	0	0	0	1	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

WRITE\_ALL command write data to all the display registers simultaneously, can be used to easily turn all the segments on or off, or to display same contents on all digits.

## 7-segment Display Related Commands

### Decoded Write Command: *DECODE\_WT* (0x80 – 0x9F)

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	0	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	-	-	-	-	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This command translate data into 7-segment maps and write it to display registers. In the command byte, A<sub>4</sub>:A<sub>0</sub> is the target digit position, which is, the register address, D<sub>3</sub>:D<sub>0</sub> in data byte is the value to be displayed as 7-segment number.

D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	D <sub>3</sub> :D <sub>0</sub> (hexadecimal)	Display
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4

D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	D <sub>3</sub> :D <sub>0</sub> (hexadecimal)	Display
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	A	A
1	0	1	1	B	b
1	1	0	0	C	C
1	1	0	1	D	d
1	1	1	0	E	E
1	1	1	1	F	F

(Table 6: Data Value vs 7-segment Display)

DECODE\_WT command will only update lower 7 bits in the display registers, the highest bit (DP bit) will remain unchanged, so it is both easier for using these segments as decimal points and as extra digits.

There is an exception, when the 2nd byte is 0x80, instead of displaying '0' according to Table 6, BC7591 will give a blank display. This feature can be used to eliminate the leading '0's when needed (such as when displaying decimal numbers), or be used to clear some displaying digits while keeping extended digits untouched.

### Write Extended Digits Command: WRITE\_EXT(0xA8 – 0xAB)

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	1	0	1	0	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This command acts like the direct register write command, but the target is not a display register, instead it will write the data into bit7 of each display register, which can be used to drive extended display digits. 32 display registers can be divided into 4 groups, each with 8 registers, the A<sub>1</sub>:A<sub>0</sub> in command byte will be treated as group number, then D<sub>7</sub>:D<sub>0</sub> will be written to the bit7 of each register in that group accordingly.

This command is an alias of the QTR\_WT\_TOP command.

## Decoded Write To Extended Digit Command: *DECODE\_EXT (0xB0 – 0xB3)*

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	1	1	0	0	A <sub>1</sub>	A <sub>0</sub>	-	-	-	-	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This write command has the same target as WRITE\_EXT command, but the data written is decoded 7-segment map as in Table 6. It's dedicated to be used with extended 7-segment display digits. Just like the DECODE\_WT command, it will only update lower 7 bits of the extended digit, and it will display 'blank' if the 2nd byte is 0x80.

## Dot Matrix Display Related Commands

### Column Write Command: *COL\_WRITE (0x00 - 0x1F)*

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	0	0	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

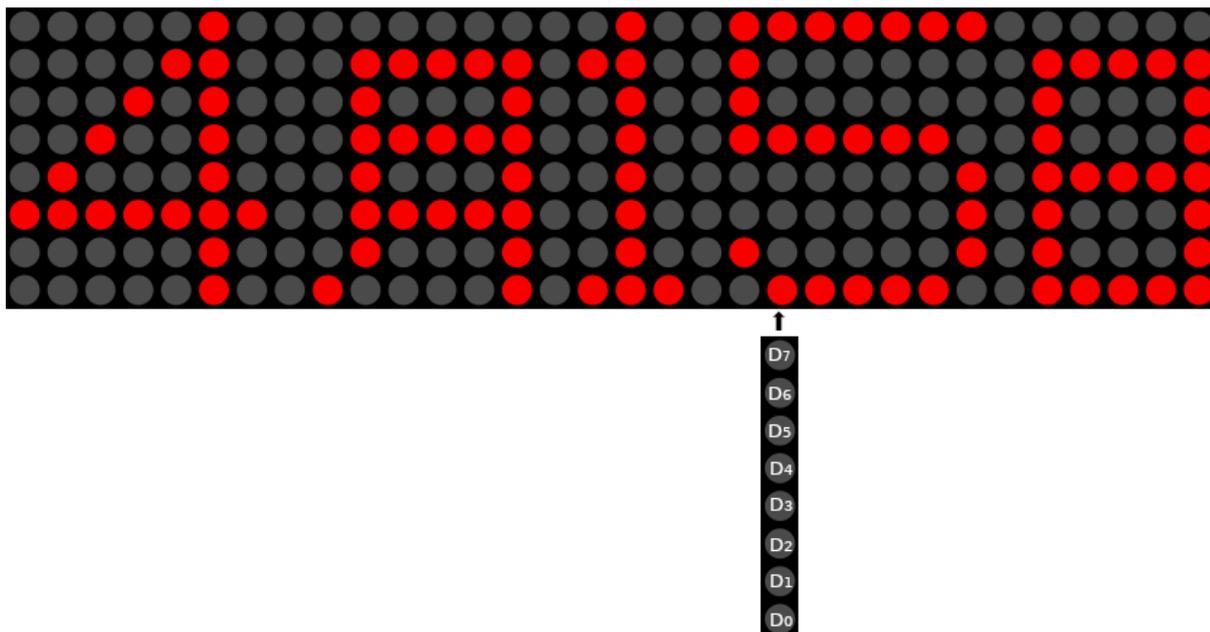
This is an alias of DIRECT\_WT command. When using an dot matrix display, each display register represents a column of the matrix, the name column write makes it easier to be understood than register write.

### Insert And Shift High Write Command: *SHIFT\_H\_WT (0x40 – 0x5E)*

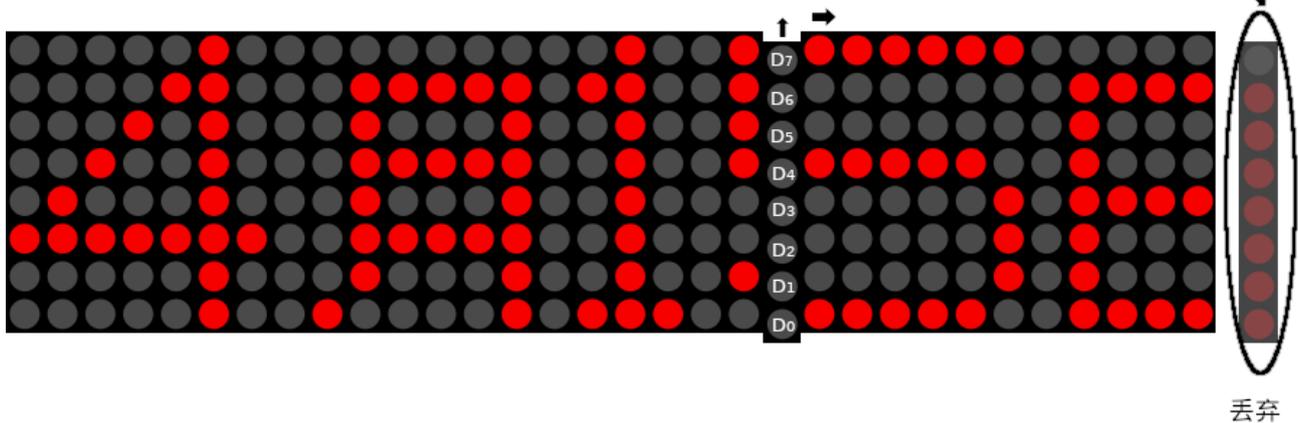
1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	1	0	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This command will insert data at column A<sub>4</sub>:A<sub>0</sub> and shift the existing content to the higher column (register) direction. A<sub>4</sub>:A<sub>0</sub> is the column number, which is also the register address, the range is 0x00-0x1E.

The following diagrams illustrate how this command works:



(Diagram 7: Before Command Execution)



(Diagram 8: After Command Execution)

**Insert And Shift Low Write Command: SHIFT\_L\_WT (0x61 – 0x7F)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	1	1	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

Similar with the previous one, but this command will shift to lower column(register) direction. In this command, the address range is 0x01-0x1F.

**Rotate Right Command: ROTATE\_R(0x5F)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	1	0	1	1	1	1	1	-	-	-	-	-	-	-	-

Assuming the lower column is on left, this command rotate the display content right (high column direction), the most right (highest) column rotate back to column 0. The second byte can be arbitrary data.

**Rotate Left Command: ROTATE\_L(0x60)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	1	1	0	0	0	0	0	-	-	-	-	-	-	-	-

This command is similar to the ROTATE\_R but with a opposite rotate direction.

**Write Quarter Line At Bottom Command: QTR\_WT\_BOT (0xA0 – 0xA3)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	1	0	0	0	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

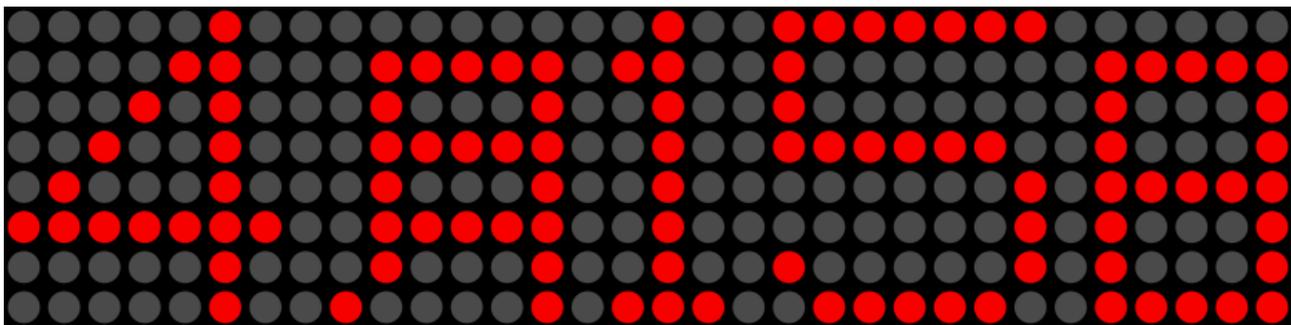
When the display forms a 32x8 dot matrix, every column is a display register, therefore write by column is the most natural way to update the content. But BC7591 also provides a way to write by row. This command update ¼ of the bottom line a time. Again, the A<sub>1</sub>:A<sub>0</sub> is the group address of the column groups, the concept is same as in the WRITE\_EXT command.

### Insert Quarter Line At Bottom Command: QTR\_INS\_BOT (0xA04– 0xA7)

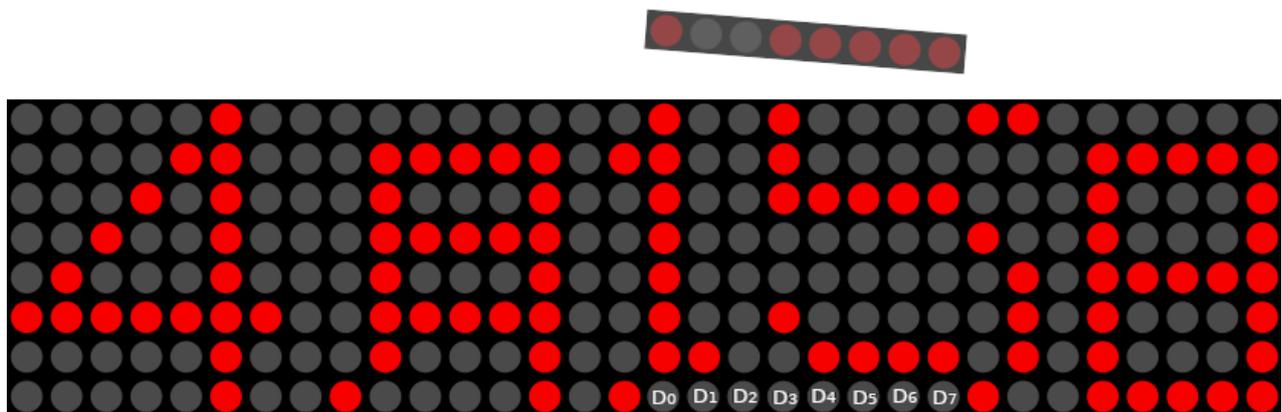
1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	1	0	0	1	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

Similar to the previous command, but this command will do insert instead of overwrite.

The following diagram illustrate how this command works:



(Diagram 9: Before Command Execution)



(Diagram 10: After Command Execution)

From the register's point of view, this command shift data of all the display registers in the selected group left and then write the bits in the data byte to bit0 of each register.

### Write Quarter Line At Top Command: QTR\_WT\_TOP (0xA8 – 0xAB)

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	1	0	1	0	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This is an alias of the WRITE\_EXT command. It is similar to the previous command but will write into the top line of the dot matrix. When using 7-segment displays with extended digits, this command is used as "write display register" command of the extended digits.

### **Insert Quarter Line At Top Command: QTR\_INS\_TOP (0xAC – 0xAF)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	0	1	0	1	1	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This command acts similar to the previous QTR\_WT\_TOP command but use an insert action and shift the existing display downwards.

### **Coordinate Write Command: COORD\_OFF, COORD\_ON (0xC0, 0xC1)**

Coordinate Off (Write 0) Command 0xC0:

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	0	0	0	0	0	0	X <sub>4</sub>	X <sub>3</sub>	X <sub>2</sub>	X <sub>1</sub>	X <sub>0</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>

Coordinate On (Write 1) Command 0xC1:

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	0	0	0	0	0	1	X <sub>4</sub>	X <sub>3</sub>	X <sub>2</sub>	X <sub>1</sub>	X <sub>0</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>

These 2 commands are same as the segment access commands SEG\_OFF and SEG\_ON. When the display is considered a 32\*8 matrix, every dot in the matrix has a coordinate (x, y), where x can range from 0 to 31 and y will have a range of 0-7. If we look at the segment address of each LED in the matrix, the lower 3 bits is just the same as its y coordinate, and the higher 5 bits is same as the x.

## **Controlling Commands**

### **Blink Control Write Set Command: BLINK\_WT\_SET (0x30 – 0x3F)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	0	1	1	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

BC7591 supports individual blinking for the first 128 segments, in other words, the first 16 display registers. In this command, A3:A0 is the register address, data byte maps to the segments in display register. The segments mapped by 1s will have its blinking enabled, while the 0s will keep its blink status unchanged.

The blinking is also controlled by the global blink control command. The blink feature is independent to the display register content, once the blink is enabled, even if the display content is cleared, the blink status will remain and the segment will start to blink when the register bit is set to 1 again. All blink status will be reset at power on.

### **Blink Control Write Clear Command: BLINK\_WT\_CLR (0x20 – 0x2F)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	0	1	0	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

This command acts similar to the BLINK\_WT\_SET, except the '1' bits in this command's data byte will clear the blink status of the corresponding segments.

### Digit Blink Control Command: **BLINK\_DIG\_CTL (0xD0, 0xD1)**

Digit 16 to 23 blink control command 0xD0 (for display register address 0x10 to 0x17) :

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	0	1	0	0	0	0	D <sub>17</sub>	D <sub>16</sub>	D <sub>15</sub>	D <sub>14</sub>	D <sub>13</sub>	D <sub>12</sub>	D <sub>11</sub>	D <sub>10</sub>

Digit 24 to 31 blink control command 0xD1 (for display register address 0x18 to 0x1F) :

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	0	1	0	0	0	1	D <sub>1F</sub>	D <sub>1E</sub>	D <sub>1D</sub>	D <sub>1C</sub>	D <sub>1B</sub>	D <sub>1A</sub>	D <sub>19</sub>	D <sub>18</sub>

BC7591 does not support individual segment blink for display register 0x10 to 0x1F, blink is only controlled at register range. In these 2 commands' data bytes, D<sub>n</sub> stands for the control of Digit n, for example D<sub>12</sub> bit will control the blinking of digit 12. When the bit is 1, the represented digit will have its blink on.

After a power on reset, all the digits from 0x10 to 0x1F will have its blink status cleared.

### Blink Speed Control Command: **BLINK\_SPEED (0xF2)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	1	1	0	0	1	0	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

Used for blink frequency control, the frequency can be roughly calculated by

$$F_{\text{blink}} = 92/(2*S)$$

where S is the data byte. After a power-on reset, the internal S value is reset to 50, or the blink frequency is about 0.92Hz.

### Dim Control Command: **DIM\_CTL (0xF3)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	1	1	0	0	1	1	-	-	-	-	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>

BC7591 has 16 dim levels, The second byte of the command is the new dim level to be set, only the lower 4 bits are used and the other bits will be ignored. The display will be in its brightest state when the level is 0, and darkest for 0x0F.

### Global Control Command: **GLOBAL\_CTL (0xF0)**

1st byte(command)								2nd byte(data)								
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>		d <sub>0</sub>
1	1	1	1	0	0	0	0	r	r	r	r	r	r	DISP_OFF		BLINK_OFF

This command controls 2 global properties of the BC7591: the global blink on/off and global display on/off. The least significant 2 bits in data byte are control bits, others are reserved, should be set 0 when using this command.

bit0 BLINK\_OFF: If set 1, the blink function will be disabled. This control has higher priority than the individual segment and digit blink controls, but individual settings will remain and take control after this bit is set 0 again.

bit 1 DISP\_OFF: Set 1 to turn off the whole display output, while keeping the key scan function. The contents will NOT be cleared by setting this bit.

### **Reset Command: RESET (0xFF5A)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0

After receiving this command, the BC7591 will do a reset back to the state same as the power-on reset. If there is any key kept pressed during this command, BC7591 will output all of the pressed key values after reset. See keyboard section of this document for more information.

### **UART Send 0 Command: UART\_SEND\_0(0xFFFF)**

1st byte(command)								2nd byte(data)							
d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Forcing the chip to output a single byte of 0x00 through its Tx. This can be used to measure the real baud rate of the BC7591, or to calibrate the host side baud rate. It's usually only needed when the controller is also clocked by a RC oscillator which will drift with temperature. See Baud Rate Error section for more information.

## Summary of BC7591 Commands

Value	Commands	Summary
0x00 - 0x1F	DIRECT_WT COL_WRITE	Direct write to display registers/columns
0x20 - 0x2F	BLINK_WT_CLR	Clear segment blink property
0x30 - 0x3F	BLINK_WT_SET	Set segment blink property
0x40 - 0x5E	SHIFT_H_WT	Insert column and shift high/right
0x5F	ROTATE_R	Rotate display right/high by 1 pixel
0x60	ROTATE_L	Rotate display left/low by 1 pixel
0x61 - 0x7F	SHIFT_L_WT	Insert column and shift low/left
0x80 - 0x9F	DECODE_WT	7-segment decoded write
0xA0 - 0xA3	QTR_WT_BOT	Write ¼ line to bottom of matrix
0xA04- 0xA7	QTR_INS_BOT	Write ¼ line to bottom of matrix and shift up
0xA8 - 0xAB	QTR_WT_TOP WRITE_EXT	Write ¼ line to top of matrix/Write to extended digits
0xAC- 0xAF	QTR_INS_TOP	Insert ¼ line to top of matrix and shift down
0xB0 - 0xB3	DECODE_EXT	7-segment decoded write to extended digits
0xC0	SEG_OFF COORD_OFF	Segment Off/Coordinate Off
0xC1	SEG_ON COORD_ON	Segment On/Coordinate On
0xD0 - 0xD1	BLINK_DIG_CTL	Digit blink control
0xF0	GLOBAL_CTL	Global control
0xF1	WRITE_ALL	Write to all display registers
0xF2	BLINK_SPEED	Blink speed control
0xF3	DIM_CTL	Dim level control
0xFF5A	RESET	Reset
0xFFFF	UART_SEND_0	Output 0x00 through UART

(Table 7: Summary of BC7591 commands)

This is all of the acceptable commands BC7591 can receive, if anything out of this range is sent to BC7591, the reaction of the chip will be undefined.

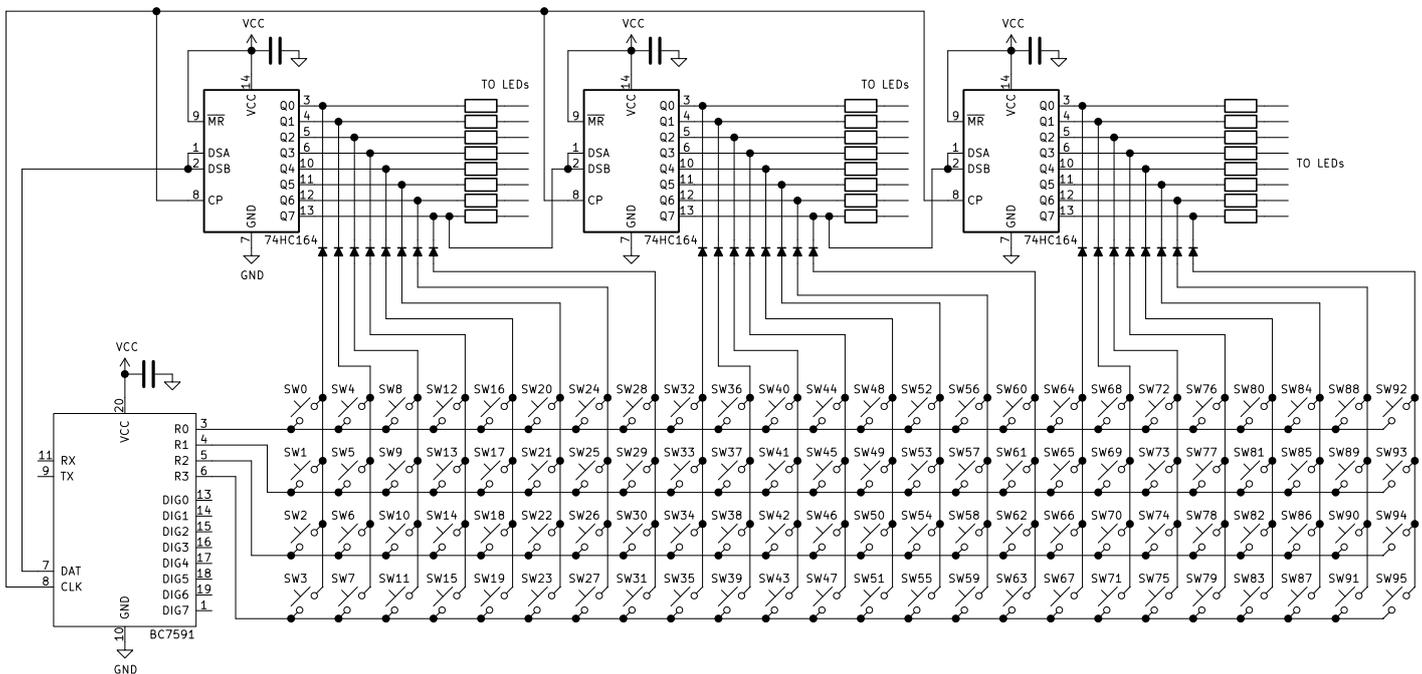
There will be a 300ms period of time between the power-up and any chip operation, this is designed to let the controller get into a working state can receive the output from BC7591, during this period, BC7591 will not accept any command either.

## Keyboard Interface

### Circuits

Besides the LED driver, BC7591 provides a keyboard interface too. It has 4 key matrix scan lines R0-R3, which will be the rows, and the key matrix columns will be connected to the outputs of the shift registers. According to the number of shift registers used, the matrix size can be from 32 keys (4\*8) to 96 (4\*24). The keyboard interface supports any types of key combinations, and long-press of any length of time. It supports normally open or normally closed type of switches, or a mixture of these two types.

The following diagram is the complete circuit for a 96-key matrix, display parts not included. The annotation number of the switches also stand for its key values, for example SW0 will have a key value of 0, and SW25's key value will be 25(0x19).



(Diagram11: Keyboard Circuits)

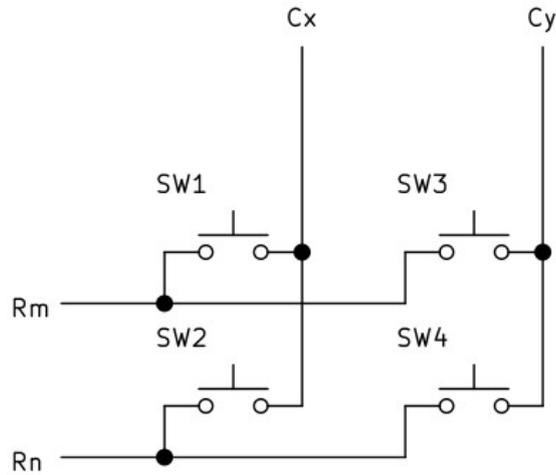
As keyboard and LEDs are sharing the output lines of the shift registers, to prevent the short circuit caused by simultaneously pressed switches between these outputs, diodes must be used to isolate these lines. Low forward voltage diodes such as Schottkies should be used to keep key scanning working reliably. This is especially important when the chip is running at a lower supply voltage. When VCC is 3.3V, the highest  $V_{IL}$  on R0-R3 will be 0.7V, similar to a standard forward voltage drop on a diode, if a normal diode is used as the isolation diode, the key scan may not be working properly.

If anti key-ghosting diodes have already been used (see next section), these diodes can prevent the shift register outputs from being shorted as well, so the diodes in diagram 11 can be eliminated.

BC7591 has built-in pull-up resistors at R0 - R3, the equivalent resistance is about 100K $\Omega$ , so there is no need to use external pull-ups in typical applications. If the keyboard is spreading in a large area, then noise level could be increased, additional pull-up resistors could be needed to lower the input impedance, but the pull-ups should not be too low otherwise there will be significant amount of current flow in the LEDs when the key is pressed.

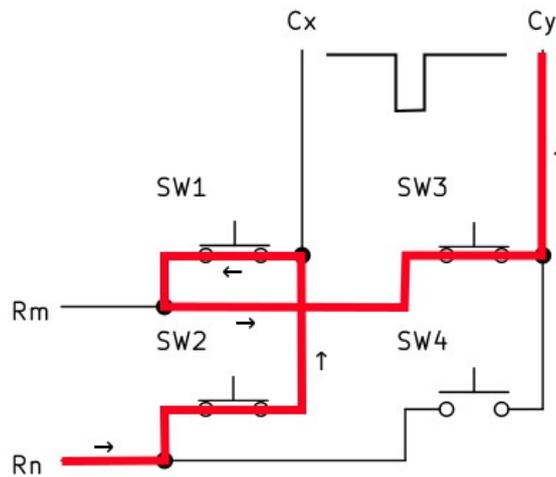
## Prevent Key Ghosting

BC7591 is capable to handle any combination of keys, but when the combination is of 3 keys at 3 corners of a rectangle, special cares must be taken to prevent ghost keys. Key ghosting is a phenomenon for a key matrix, when 3 keys are located at 3 corners of a rectangle, pressing all the 3 keys together will make the 4th key on the left corner seems being pressed too. In diagram 12, when SW1, SW2 and SW3 are both pressed, the SW4 will seem to be pressed too.



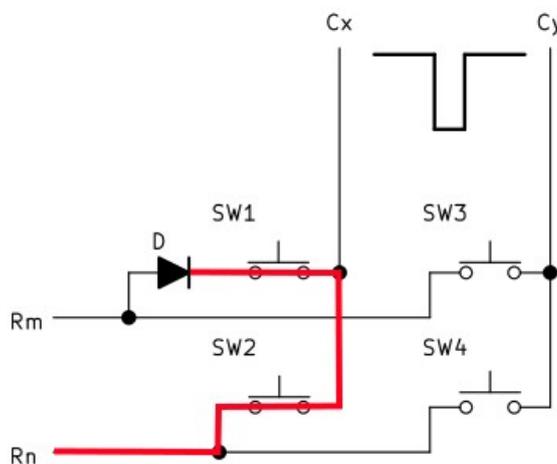
(Diagram 12: Keys located at 4 corners of a rectangle)

The reason behind this is when all the SW1-SW3 are connected, the 2 pins on SW4 will be connected too, through a path outside the SW4, like shown in diagram 13, this will let the key scanner think the SW4 is in a conducted state.



(Diagram 13: When 3 keys are pressed, the 4th one will seem to be pressed too)

To solve this problem, a diode can be added in series to SW1. The diode will cut the path of current flow, so when the scanner put a low at Cy, the Rn will not be pulled down then SW4 will not be seen as pressed.



(Diagram 14: Using A Diode To Prevent Key Ghosting)

To prevent ghosting on all keys, need diodes to be added to every key in the matrix, this will increase the cost significantly when the number of keys scales up. In real world application, it's common only a few important keys are diode protected. These anti-ghosting diodes should have as low voltage drops as possible, Schottky diodes are preferred.

## Output Format

The output format of BC7591 is same as the BC6xxx series keyboard interface chip, code can be interchangeable. When a key state changes, such as pressed(ON) or released(OFF), BC7591 will send its value, the key value is same as the switch number as shown in the Diagram 11. If the key switch is changing from OFF(released) to ON(pressed), then the original key value will be output, if it's from ON to OFF, then the bit7 of the the output data will be set. For example, if the sw1 in Diagram 11 is pressed , it will output 0x01, then when sw1 is released, the output will be 0x81. If multiple keys are changed simultaneously, all the key values will be output sequentially, keys in same column but at lower rows will be output before those at higher rows, but which column will be the first output will depends on the key scan status at the moment.

After power on, the default state of all keys are in OFF state. If any key was already in ON state at the moment of power on, BC7591 will output all the key values of those are ON. In an extreme situation, if all the keys are all in ON condition, all the key values will be output at power on.

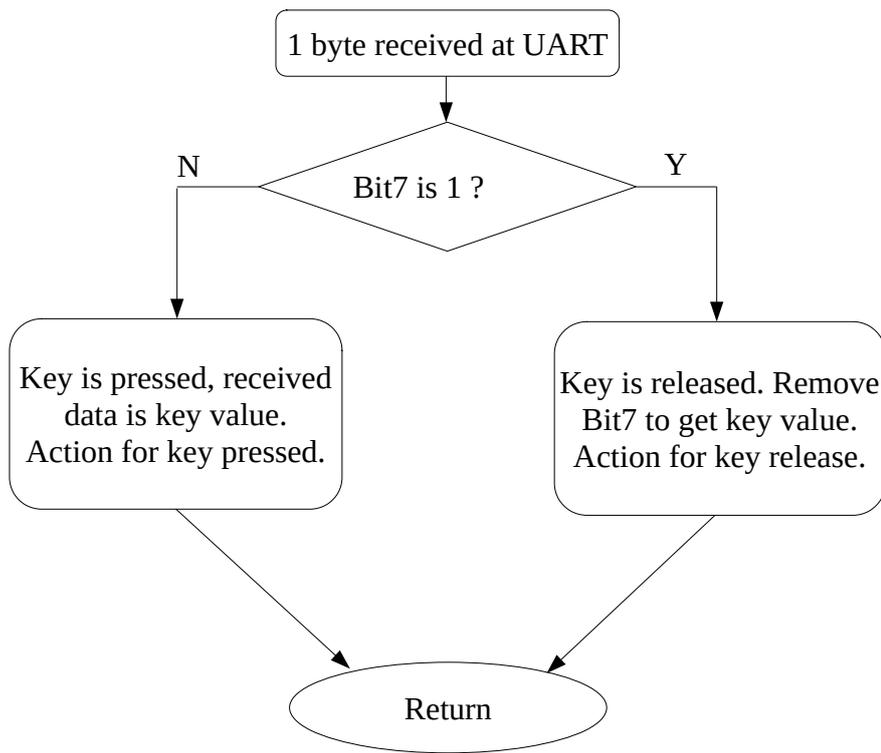
Between the power on and the beginning of key scan, there is a delay of 300ms, this is to ensure the receiving processor can be in fully operational condition for any possible data sent from BC7591.

## Processing Flow Chart

Usually in MCU there will be an interrupt associated with the UART, it's a best practice to use the interrupt to process the communication between MCU and BC7591. Since the key value will be output directly, it's very simple for the MCU to process the data, in a simple application, what needed to do is just jumping to different routines according to the received value.

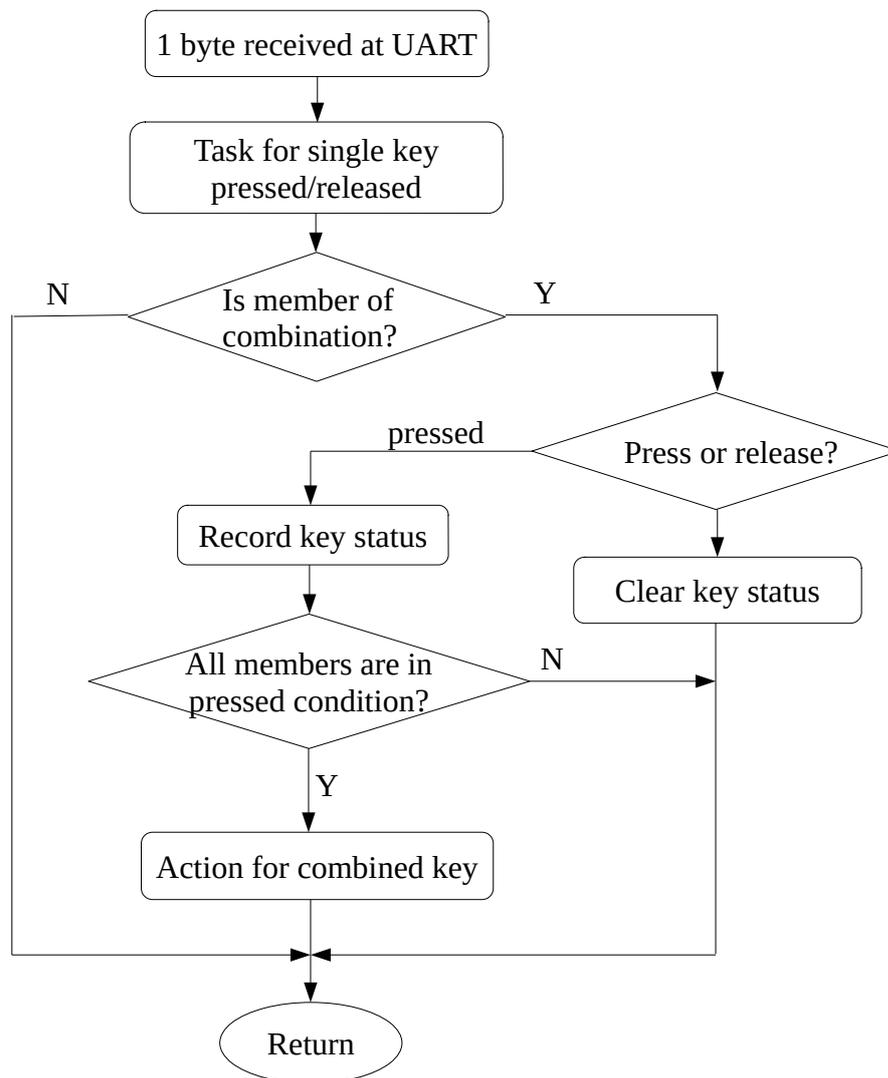
When multiple key events are happened at same time, BC7591 will output all the key values continuously. The baud rate is fixed 9600, so the time between 2 data will be about 1ms. The MCU is expected to finish the key processing within 1ms. When the processing time is longer than 1ms, a UART buffer should be used to prevent data lost.

## Flow Chart For Single Key Event



(Diagram15: Single Key Event)

## Key Combinations



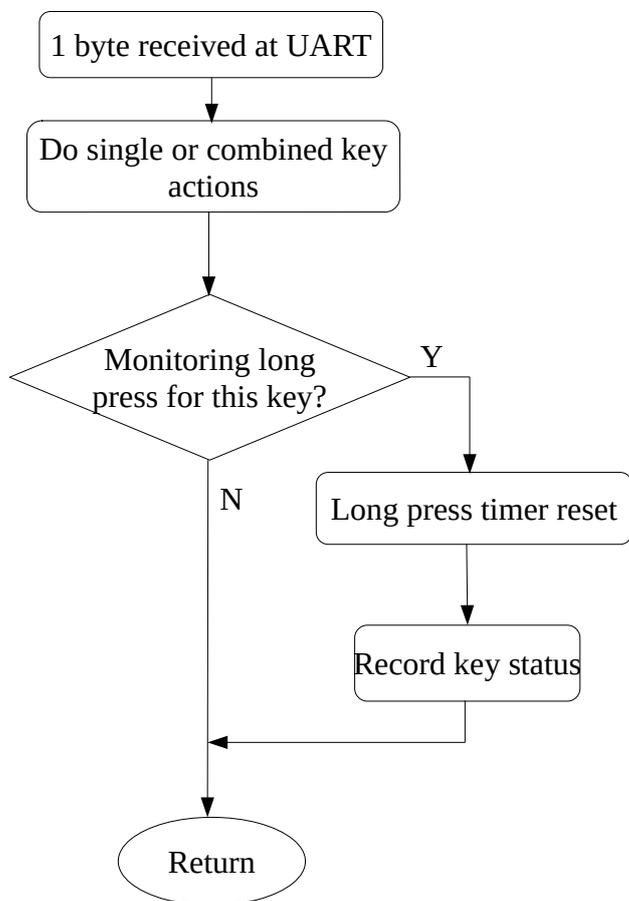
(Diagram16: Key Combinations)

It is required the MCU can remember the status of each member of the key combination.

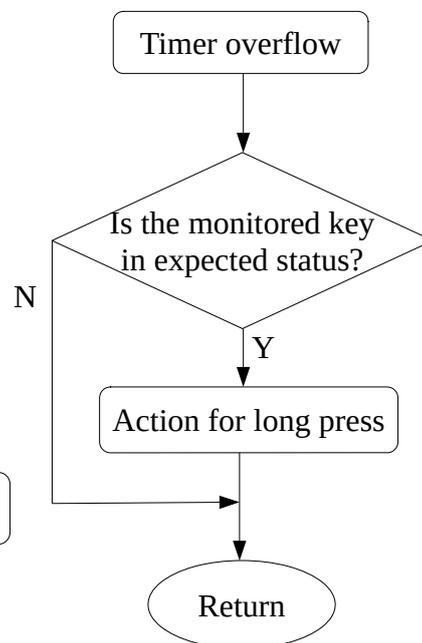
## Long Press

BC7591 will output key value whenever the status of the keyboard changes, in other words, if there is no output, it means the keyboard keeps in its current status, therefore when a key is pressed and there's no data output for a while, we can know that key is keeping pressed for that period of time. Not only for single key press, the key combination, even the key release can be detected with this method. The user will need a timer to detect a certain length of time.

UART interrupt:



Timer:



(Diagram17: Long Press)

## UART

### Settings

The UART in BC7591 has the following parameters: baud rate 9600, 8 data bits, 1 start bit, 1 stop bit, no priority check.

### Error Detection

There are 2 ways for BC7591 to detect errors in communication. Firstly, for the received data, if there were frame errors, the data is discarded.

Besides the frame error detection, BC7591 has a unique deadline-reset scheme to prevent communication errors. Each command for BC7591 is composed of 2 bytes of data, first byte for command and the second is data. If for any reason, the second byte is not coming within a particular length of time, the 1st one will be discarded. The deadline for the second byte is 2 display scan cycles.

BC7591 has a display scan frequency of about 93Hz, so one cycle is about 10.8ms. Therefore if the time between 2 commands is larger than 21.6ms, the commands can be guaranteed to be synchronised, data bytes will not be treated as

command byte even if the command byte is lost. There is no need to keep this time interval between every consequent commands.

## Interface Circuit

### I/O Levels

The Tx on BC7591 is open drain, it can be easily connected to MCUs with different VCC, only a pull-up resistor is needed at the MCU end. Many MCUs and UART chips has pull-up resistors built-in, in these case the external resistor may not be used.

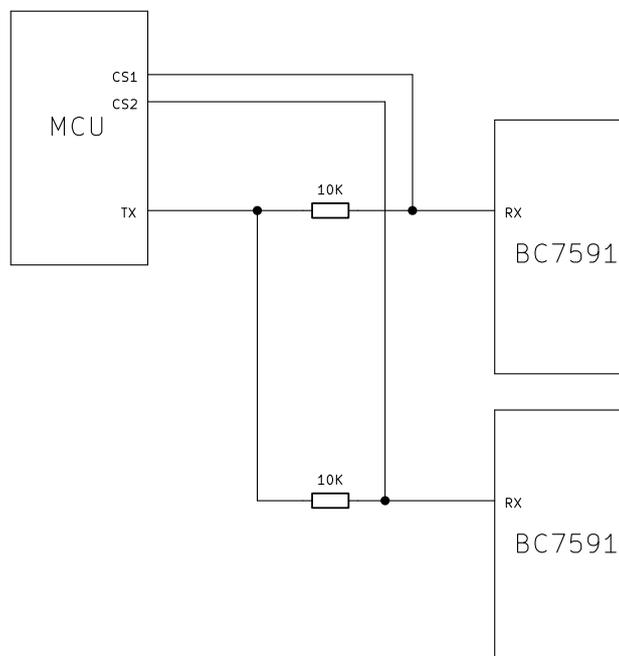
The Rx port has an internal pull-up resistor. If the power supply voltage are different between BC7591 and micro controller, special considerations should be taken. If the output voltage from MCU is higher than the VCC on BC7591, and the output is a push-pull high current drive, a proper resistor should be placed between the controller and BC7591 to prevent damage to the Rx port on BC7591. 10k should be good for 5V-3.3V situations in general.

If the output voltage is less than the VCC on BC7591, meeting the requirement of minimum high level voltage  $V_{IH}$  of Rx must be considered. The  $V_{IH}$  on Rx pin is  $0.8V_{CC}$ , so if the VCC on BC7591 is 5V, the minimum high level on Rx will be 4V, higher than most of the 3V/3.3V system output.

### Multiple Chips

UART port is usually used for 1 on 1 communication, but if more than 1 BC7591 is needed in a system, a simple circuit can be used as 'CS' signal so all the BC7591s can be driven by a single UART port. Display commands are always sent through the Tx port on micro controllers, and Rx port on controller is for receiving the keyboard output. Normally a single BC7591 can provide enough keys so receiving form multiple BC7591s will not be necessary.

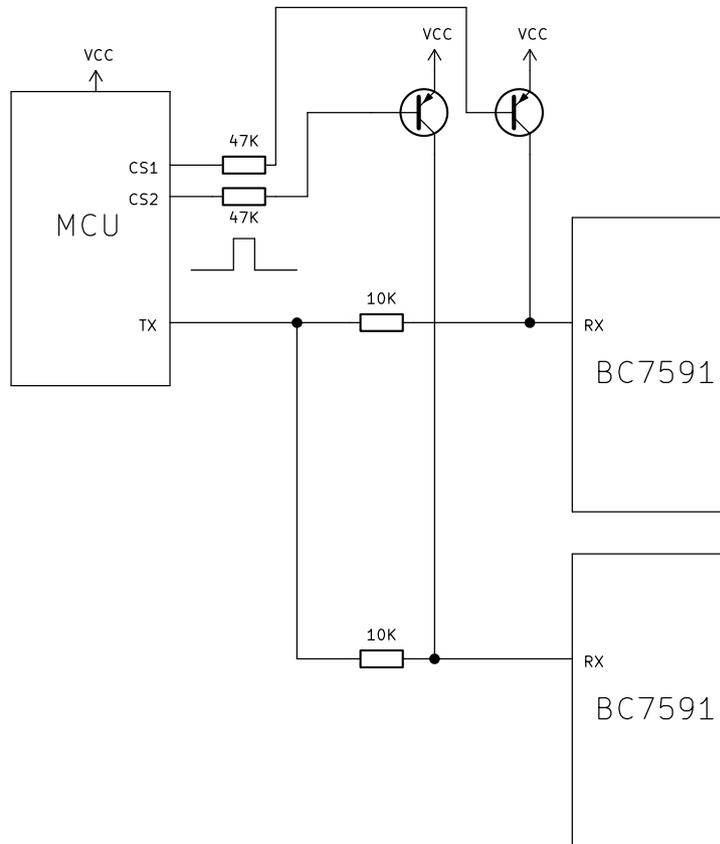
If the CS output on micro controller support both output and hi-z status, a single resistor will create the CS circuit as in the following diagram:



(Diagram18: Using I/O port as a CS for UART)

For those not selected lines, the CS<sub>n</sub> will be set to output and driven high, so whatever the Tx level is, the connected Rx pin on BC7591 will always be high (idle). To 'select' a BC7591, set the corresponding CS<sub>n</sub> to high-impedance (input) mode, so the level on the Rx pin will follow the change on Tx pin.

If CS pins don't support input mode, the circuit will be a little bit complex, 2 transistors are needed as below:



(Diagram19: Using transistors to create CS circuit)

When CSn pin outputs low, the transistor will be on, so Rx pin on BC7591 will be held high, when CSn is high, the transistor will be off, then Rx will be free to follow the level on Tx.

## Baud Rate Error

BC7591 uses an internal calibrated RC oscillator to generate the baud rate. In whole working temperature and voltage range, the frequency error is less than  $\pm 1.5\%$ . UART protocol requires the relative frequency error must be less than  $\pm 5\%$  between the 2 sides, so the baud rate error on micro controller must be less than  $\pm 3.5\%$ .

If the baud rate can be fine tuned on the controller, by measuring the rate on BC7591, the controller can tune and minimise the the relative error. The simplest way to measure the baud rate is to measure the pulse width of a 0x00 byte. In theory, when the baud rate is 9600, a 0 output is a negative pulse of 937.5us width. Calculate the pulse width can get the error rate of BC7591.

To let BC7591 output 0, there are 2 methods. First is to press the key number 0, to get an output 0 by programming, send the 0xff 0xff command, then BC7591 will output a 0 too.

## Absolute Rating

Storage Temperature	-65°C - +150°C
Working Temperature	-40°C - +85°C
Voltage On Any Pin (Ext. Tx)	-0.5 - +6.0V

(表 8: 极限参数)

## Electrical Characters

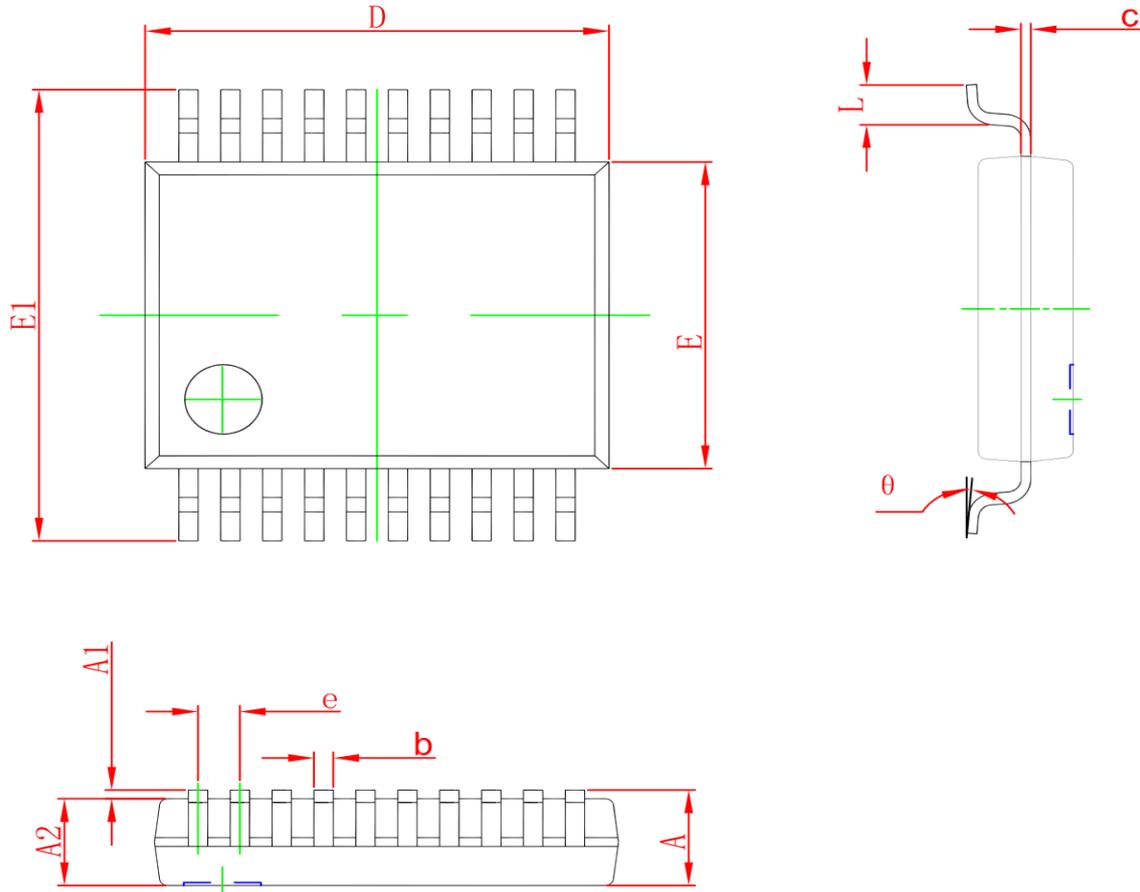
(Note:  $T_A=25^{\circ}\text{C}$  unless notified)

Parameter	Min	Typical	Max	Comments
$V_{CC}$ Working Voltage	3.0V		5.5V	
$I_{CC}$ Working Current		1.45mA		$V_{CC}=3.3\text{V}$ , 无外接电路
		2.75mA		$V_{CC}=5.0\text{V}$ , 无外接电路
$\Delta_F$ Frequency Drift			$\pm 1.5\%$	$V_{CC}=3.3\text{V}$ , $-40^{\circ}\text{C} - +85^{\circ}\text{C}$
			$\pm 1\%$	$V_{CC}=3.3\text{V}$ , $-10^{\circ}\text{C} - +70^{\circ}\text{C}$
			$-1\%+0.5\%$	$V_{CC}=5.0\text{V}$ , $-40^{\circ}\text{C} - +85^{\circ}\text{C}$
$V_{IL}$ Input Low Level			0.7V	$V_{CC}=3.3\text{V}$
			1.0V	$V_{CC}=5.0\text{V}$
$V_{OL}$ Output Low Level		0.0V		$V_{CC}=5\text{V}$ , $I_{OL}=1\text{mA}$
		1.0V		$V_{CC}=5\text{V}$ , $I_{OL}=100\text{mA}$
		1.0V		$V_{CC}=3.3\text{V}$ , $I_{OL}=60\text{mA}$
$F_{SCN}$ Display Scan Freq.		93Hz		
$T_{CL}$ Keyboard Scan Cycle		129ms		Max response delay
$T_{db}$ Debounce Time		11.5ms		Min key hold time
$T_{pu}$ Power Up Delay		300ms		
$R_{PU}$ Row Pull-up Resistance		100K		$V_{CC}=3.3\text{V}$
		55K		$V_{CC}=5.0\text{V}$

(表 9: 电气特性)

# Package Dimensions

## SSOP20(209mil) PACKAGE OUTLINE DIMENSIONS



Symbol	Dimensions In Millimeters		Dimensions In Inches	
	Min	Max	Min	Max
A		1.730		0.068
A1	0.050	0.230	0.002	0.009
A2	1.400	1.600	0.055	0.063
b	0.220	0.380	0.009	0.015
c	0.090	0.250	0.004	0.010
D	7.000	7.400	0.276	0.291
E	5.100	5.500	0.201	0.217
E1	7.600	8.000	0.299	0.315
e	0.65(BSC)		0.026(BSC)	
L	0.550	0.950	0.022	0.037
theta	0°	8°	0°	8°

## Package Information

Order Number	Package	Unit Quantity
BC7591EC-T	tube	6600
BC7591EC-RS	reel small	1000
BC7591EC-RL	reel large	2000