An extended version of Traffic Hat for Raspberry pi experimentation.

- Eagle schematics and Gerber files.
- Basic Hat programming.
- Code examples with a 7-segment LED.
- Code examples with 8X8 dot matrix LED.

Traffic hat for Raspberry pi

John Ellinas

Athens 2017

Author Name

Traffic hat for Raspberry pi

John Ellinas

For information contact: http://android4arduino.com

First Edition: June 2017

Contents

Introduction

THis booklet is a guide to develop and experiment on the traffic hat for Raspberry pi as that shown in Fig 1. Traffic hat is a basic experimentation kit for Raspberry pi beginners who may develop Python code snippets in order to be familiar with Python language and interfacing to Raspberry pi using its GPIO. This kit has the following characteristics:

- It is equipped with one tactile button, three LEDs and one buzzer.
- It has a connector for connection to Raspberry pi GPIO.
- No software library needed for programming.

Fig 1: Traffic hat for Raspberry pi

The traffic hat developed for this post is an extended version of the original one as it is shown in Fig 2.

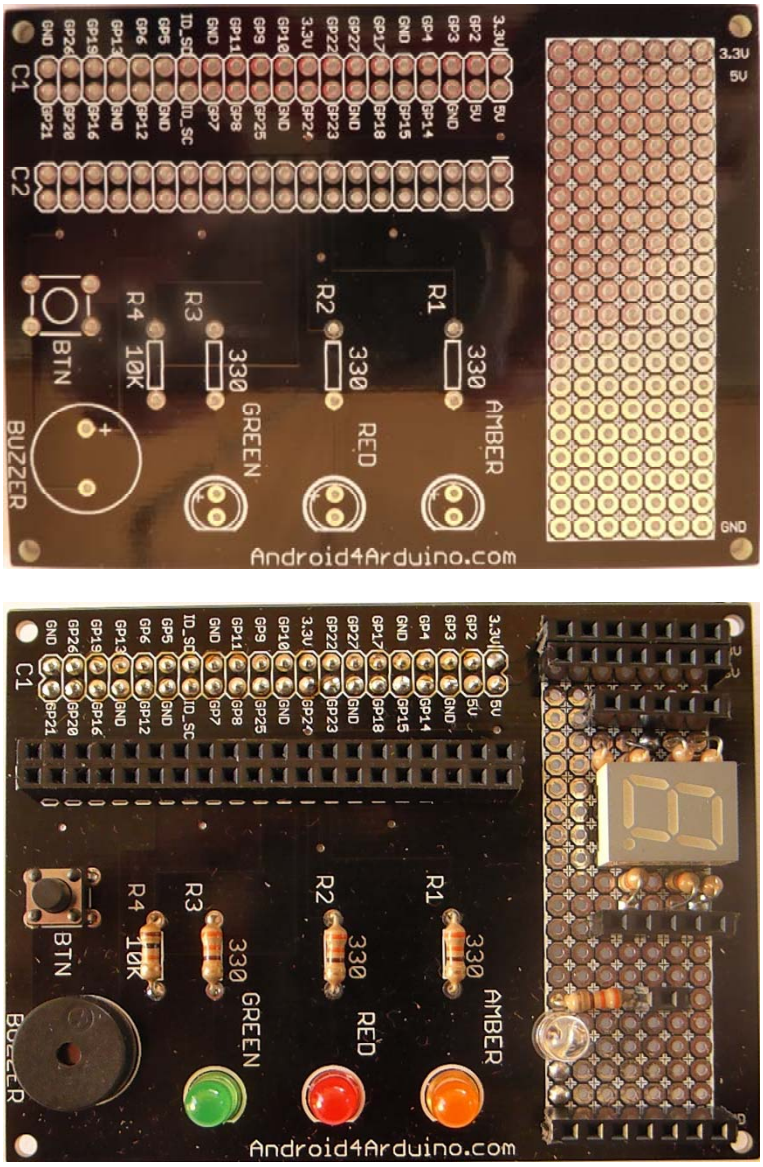


Fig 2: Extended version of Traffic hat for Raspberry pi

Traffic hat

THis post gives to the reader the schematic and board eagle files as well as the Gerber files in order to print his own the extended Traffic hat for Raspberry pi. It includes the components of the original hat but it has also the following:

- A full 40 way connector for connection to the GPIO.
- A 40 way pin to pin female header for connecting wires to tested electronic parts other than those embedded in to the board.
- A grid of pads for placing other electronic components.
- Three columns of the grid supply to the user 3.3 V, 5 V and Ground.

These features give the hat a versatile shape for Raspberry pi beginners for experimentation with other electronic parts.
In this booklet, it will be explained the operation of this kit and some python code snippets will be given for the beginners. Also, a 7-segment LED and an 8X8 LED dot matrix will be connected to the grid of pads and the python code for driving will be explained. Fig 3 shows the circuit diagram of the traffic hat components and the raspberry pi. The connections are the following:

- GPIO pin 16 is connected to the Amber LED.
- GPIO pin 18 is connected to the Red LED.

- GPIO pin 15 is connected to the Green LED.
- GPIO pin 22 is connected to the button. The button has a pull-up resistor of 10 KΩ.
- GPIO pin 29 is connected to the buzzer.

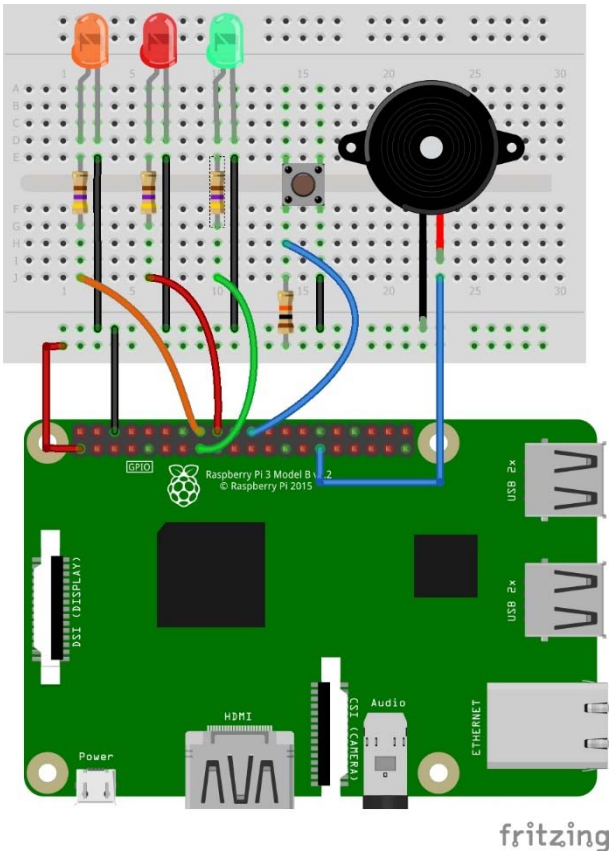
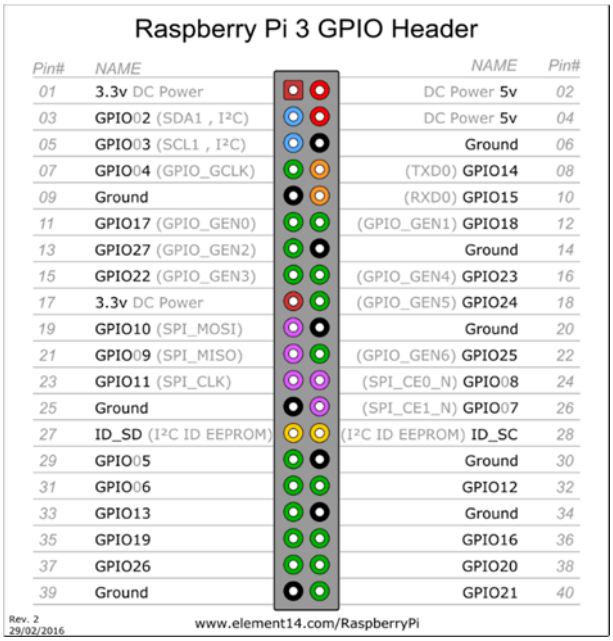
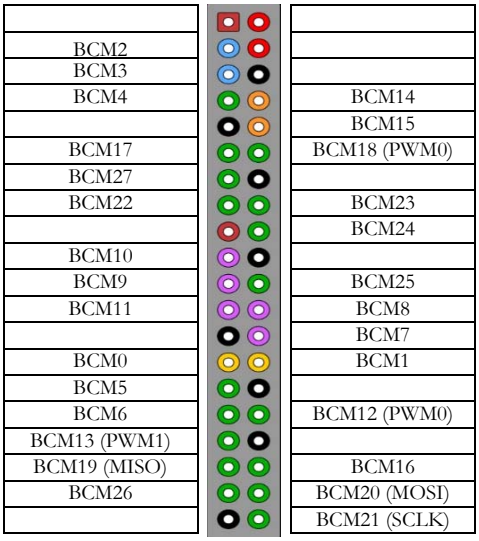


Fig 3: Raspberry pi connection with Traffic Hat components.

The GPIO header pinout is shown in Fig 4 where all pins are numbered and their alternative functions are depicted either as Pi pin header numbers (a) or as Broadcom pin numbers (b). It is apparent that GPIO gives the user access not only to the bi-directional I/O pins, but also to Serial port (UART), I2C, SPI, PWM (for analog output).



(a)



(b)

Fig 3: (a) GPIO header pinout.
(b) BCM alternative pinout.

We must always have in mind the following observations when we are trying to connect components to GPIO.

- Make any connections when Raspberry pi is switched off.
- Do not connect any GPIO input pin with voltage more than 3.3V.
- GPIO output pins cannot source more than 3mA. For LEDs, 3mA is enough to light a red LED with a 470 Ω limiting resistor.
- Do not power the Pi with more than 5V.
- Do not draw more than a total of 50mA from the 3.3V supply pins.
- Do not draw more than a total of 250mA from the 5V supply pins.

Basically, input signals that are in the scale [0, +5V] can be converted to scale [0, +3.3V] either using a level shifter IC or with a simple resistor divider, as Fig 4 shows.

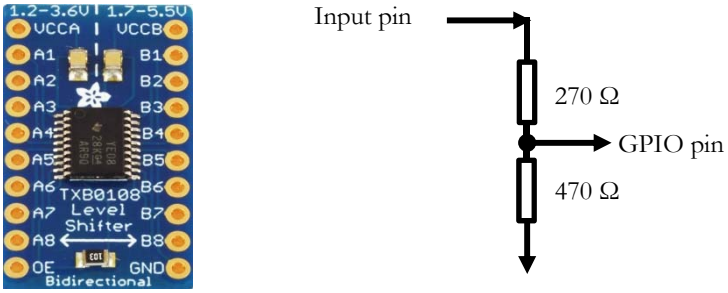


Fig 4: Voltage level conversion.

GPIO library

The Python library for driving GPIO is called RPi.GPIO and is included in Raspbian distribution. The steps for using this library in a Python snippet and the basic input-output commands are:

- Import the library and make a reference to the GPIO object:

```
import  RPi.GPIO as GPIO
```

- Declare the pin numbering scheme following the header pinout or the Broadcom chip pinout, as mentioned above:

```
GPIO.setmode(GPIO.BOARD) or
GPIO.setmode(GPIO.BCM)
```

- Set the operation mode of a pin as input or output:

```
GPIO.setup(pin, GPIO.IN or GPIO.OUT)
```

- Output to a digital pin:

```
GPIO.output(pin, GPIO.LOW or GPIO.HIGH)
```

LOW is equivalent to "0" or "False" and HIGH is equivalent to "1" or "True".

- Input from a digital pin:

```
GPIO.input(pin)
```

which returns "False" when input is 0V and "True" when input is +3.3V.

- The input pins may be connected to internal pull-up or pull-down resistors:

```
GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) or
GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

- The output pin that can create analog voltage in the form of PWM is pin 12 (PWM0). Firstly, we instantiate a variable with the frequency of PWM and then we output PWM signal defining the Duty Cycle:

```
pwm=GPIO.PWM(pin, frequency)
pwm.start(% Duty Cycle)
```

To change the Duty Cycle:

```
pwm.ChangeDutyCycle(% Duty Cycle)
```

To stop PWM output:

```
pwm.stop()
```

- To create delays, we should use the library "time":

```
include time
time.sleep(delay in seconds)
```

Example-1 on the Traffic Hat

Make a program that when button is fully pressed and released will light the LEDs and buzzer for 5 seconds. With CTRL+C there will be an exit from the program loop.

```
1  import Rpi.GPIO as GPIO
2  import time
3
4  #pin definitions
5  btnPin=22
6  ledAmber=16
7  ledRed=18
8  ledGreen=15
9  buzzerPin=29
10
11 GPIO.setmode(GPIO.BOARD)
12 #Button has an external pull-up resistor
13 #GPIO.setup(btnPin, GPIO.IN)
14 #Button has no external pull-up resistor
15 GPIO.setup(btnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
16 GPIO.setup(ledAmber, GPIO.OUT)
17 GPIO.setup(ledRed, GPIO.OUT)
18 GPIO.setup(ledGreen, GPIO.OUT)
19 GPIO.setup(buzzerPin, GPIO.OUT)
20
21 #initial state of outputs
22 GPIO.output(ledAmber, GPIO.LOW)
23 GPIO.output(ledRed, GPIO.LOW)
24 GPIO.output(ledGreen, GPIO.LOW)
25 GPIO.output(buzzerPin, GPIO.LOW)
26
27 try:
28    while True:
29        print("Press button to start. CTRL+C to exit")
30        if not GPIO.input(GPIO.btnPin)
31             while not GPIO.input(GPIO.btnPin)
32                 print("release the button")
33             else:
34                 GPIO.output(ledAmber, GPIO.HIGH)
35                 GPIO.output(ledRed, GPIO.HIGH)
36                 GPIO.output(ledGreen, GPIO.HIGH)
37                 GPIO.output(buzzerPin, GPIO.HIGH)
38                 time.sleep(5)
39                 GPIO.output(ledAmber, GPIO.LOW)
40                 GPIO.output(ledRed, GPIO.LOW)
41                 GPIO.output(ledGreen, GPIO.LOW)
42                 GPIO.output(buzzerPin, GPIO.LOW)
43 except KeyboardInterrupt:
44    GPIO.cleanup()
```

- Lines 1-2 import the libraries for GPIO and delay function.
- Lines 5-9 define the used pins with board numbering.
- Line 11 sets-up board numbering.
- Lines 13-17 define the pins mode, either as inputs or outputs. Since button has an external pull-up resistor (if connected) there is no need to activate an internal pull-up. If no external resistor is connected, Line 13 will be replaced by:

```
GPIO.setup(btnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

- Lines 20-23 set the initial state of the output pins.
- Lines 25 & 41 define a try-exception case in order to be able to exit from the infinite loop using CTRL+C.
- Line 26 sets an infinite loop.
- Line 28 is examining the press of the button.
- Line 29 is examining the release of the button.
- Lines 32-40 are executed when there is a full button press (press & release) and light-up all the LEDs for 5 seconds.
- Line 42 turns everything off.

Example-2 on the Traffic Hat

Make a program that when button is pressed will light an external LED with PWM and a Duty Cycle of 100% but when button is released will light the same LED with PWM and a Duty Cycle of 10%.

Recall that PWM is supplied by pin 12 of GPIO.

```
1  import Rpi.GPIO as GPIO
2  import time
3
4  #pin definitions
5  btnPin=22
6  ledPin=12
7
8  #set PWM duty cycle
9  dc_released=10
10 dc_pressed=100
```

```
11
12 GPIO.setmode(GPIO.BOARD)
13 #Button has an external pull-up resistor
14 GPIO.setup(btnPin, GPIO.IN)
15 GPIO.setup(ledPin, GPIO.OUT)
16 pwm=GPIO.PWM(ledPin, 100)
17
18 #initial state of LED
19 pwm.start(dc_released)
20
21 print("Press button to start. CTRL+C to exit")
22 try:
23    while True:
24        if not GPIO.input(GPIO.btnPin)
25            pwm.ChangeDutyCycle(dc_pressed)
26            time.sleep(0.05)
27        else
28             pwm.ChangeDutyCycle(dc_released)
29             time.sleep(0.05)
30 except KeyboardInterrupt:
31    GPIO.cleanup()
```

- Lines 9-10 set the Duty Cycle 100% when button is pressed and 10% when button is released.
- Line 16 instantiates PWM mode for ledPin with a frequency of 100 Hz.
- Line 19 starts PWM with DC=10%.
- Lines 23-29 define an infinite loop where the button state is checked in Line 24. When button is pressed Line 25 is executed and LED lights-up with DC=100% but when button is released Line 28 is executed and LED lights-up with DC=10%.
- Lines 26 and 29 give a small delay of 50 ms for debouncing purposes.

gpiozero library

This library gives a simple interface of various external components to GPIO. One of them is the Traffic Hat. It has to be mentioned that this library uses the BCM pin reference only. More information can be found at the url:

https://gpiozero.readthedocs.io/en/stable/

The Traffic Hat class of gpiozero library can be used as follows:

- Import the class:

```
from gpiozero import TrafficHat
```

- Instantiate an object:

```
hat=TrafficHat()
```

- Wait the button to be pressed:

```
hat.button.wait_for_press()
```

- Turn ON or OFF all the output devices:

```
hat.on()
hat.off()
```

- Turn ON or OFF LEDs:

```
hat.lights.on()
hat.lights.amber.on()
hat.lights.red.on()
hat.lights.green.on()
hat.lights.off()
h.lights.amber.off()
hat.lights.red.off()
hat.lights.green.off()
```

- Turn ON or OFF buzzer:

```
hat.buzzer.on()
hat.buzzer.off()
```

- Blink the LEDs:

```
hat.lights.blink()
hat.lights.red.blink()
```

Blink function is defined as follows:

`blink(on_time=sec, off_time=sec, n=None, background=True)`

n = number of blinks (None=Forever)

background = False→return when finishes

= True→background thread for blinking

- Toggle the output devices (if it is ON make it OFF and vice-versa):

```
hat.toggle()
```

- Toggle a specific output device:

```
hat.lights.red.toggle()
hat.buzzer.toggle()
```

- Set up events on button state:

```
hat.button.when_pressed = hat.lights.blink
hat.button.when_released = hat.lights.off
```

- PWM for the LEDs of the Traffic Hat:

```
hat=TrafficHat(pwm=True)
hat.lights.amber.value=0 (off)
hat.lights.red.value=0.5 (half intensity)
hat.lights.green.value=1 (full intensity)
```

PWM function:

`PWMLED(pin, active_high=True, initial_value=0, frequency=100)`

active_high = True – The on() method turns output to HIGH
False - The on() method turns output to LOW
initial_value = Initial value of Duty Cycle (0=OFF, 1=Full ON)
frequency = PWM frequency

Example-3 on the Traffic Hat with gpiozero library

Make a program that when button is pressed the LEDs and buzzer are ON and when is released are OFF. The program will run once.

```
1  from gpiozero import TrafficHat
2  import time
3  from signal import pause
4
5  hat=TrafficHat()
6
7  hat.button.wait_for_press()
8  hat.lights.on()
9  hat.buzzer.on()
10 hat.button.wait_for_release()
11 hat.lights.off()
12 hat.buzzer.off()
13
14 pause()
```

- Line 1 calls TrafficHat class from gpiozero library.
- Line 3 imports pause() method.
- Line 5 makes an instance of the Traffic Hat.
- Line 7 checks when button will be pressed.
- Lines 8 and 9 turn on all the LEDs and the buzzer.
- Line 10 checks when button will be released.
- Lines 11 and 12 turn off the LEDs and the buzzer.

Example-4 on the Traffic Hat with gpiozero library

Make a program that when button is pressed all LEDs will light-up with full intensity (Duty Cycle of 100%) but when button is released all LEDs will light-up with small intensity (Duty Cycle of 10%).

```
1  from gpiozero import TrafficHat
2  from gpiozero import PWMLED
3  import time
4
5  hat=TrafficHat(pwm=True)
6
7  while True:
8      hat.button.wait_for_press()
9      hat.lights.amber.value(1)
10     hat.lights.red.value(1)
11     hat.lights.green.value(1)
12     time.sleep(0.05)
13     hat.button.wait_for_release()
14     hat.lights.amber.value(0.1)
15     hat.lights.amber.value(0.1)
16     hat.lights.amber.value(0.1)
17     time.sleep(0.05)
```

- Line 1 calls TrafficHat class from gpiozero library.
- Line 2 calls PWMLED class.
- Line 5 creates an instance of the Traffic Hat object. Recall that when there is a parameter "pwm=True" (default value=False), we can create instances of PWMLED objects which are the Amber, Red, Green LEDs of the hat.
- Line 8 waits for button press.
- Lines 9-11 light-up the LEDs with full intensity (DC=100%).
- Lines 13-16 light-up the LEDs with small intensity (DC=10%).

Drive a 7-segment LED

IT is known that 7-segment LEDs are either Common Anode (CA) or Common Cathode (CC). Fig 5 shows their difference. In CA we need to source current to the common anodes of the 8 internal LEDs (including decimal point) and sink current from their individual cathodes whereas in CC we need to source current to the individual anodes of the 8 internal LEDs and sink current from their common cathodes. In any case, we put in series with every LED a limiting resistor of 330 or 470 Ω in order to have a satisfactory intensity and protect the Raspberry pi pin's current capability. Note that using a single resistor in common anode or in common cathode is not correct since the intensity of the digits will vary depending on the number of segments that are ON each time (voltage drop across it will be varying).

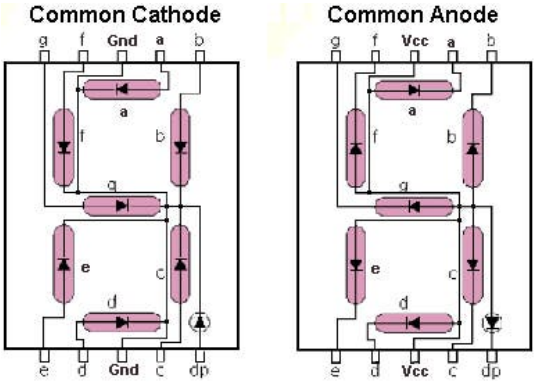
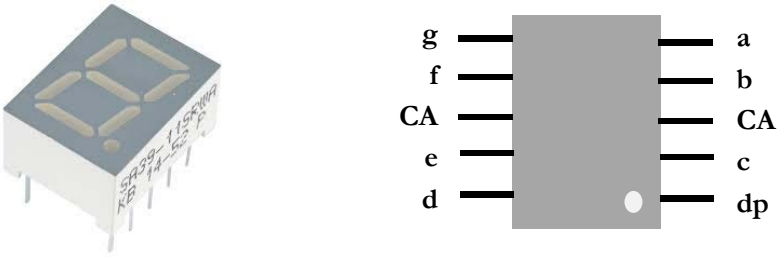


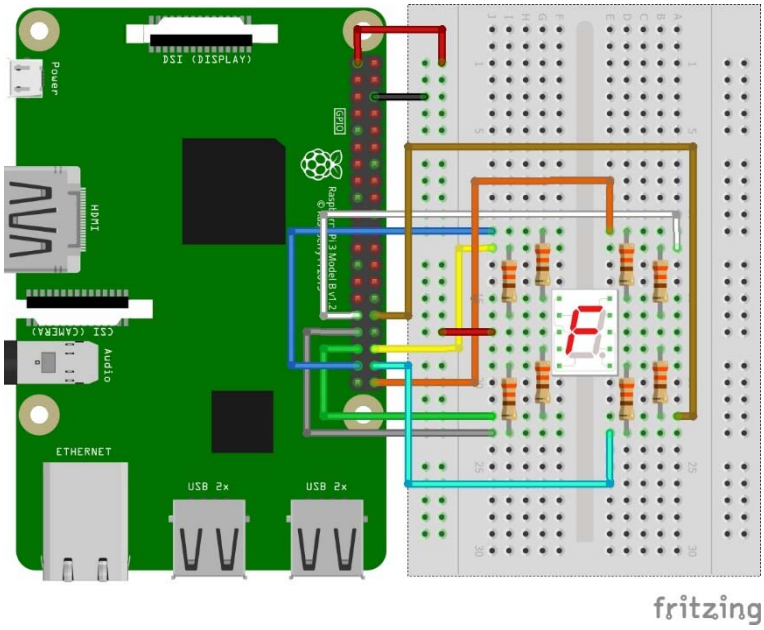
Fig 5: Common Cathode and Common Anode 7-segment LED

Fig 6(b) shows the connection diagram of a CA 7-segment LED to the Raspberry pi GPIO connector. Note that common anode is wired to +3.3V and the 8 segments are wired to different pins of GPIO. The specific CA 7-segment LED has the following pinout, Fig 6(a). The connections among segments and GPIO pins are:

a	b	c	d	e	f	g	dp
40	31	32	33	35	36	37	38



(a)



(b)

Fig 6: CA 7-segment LED connections with GPIO

The following table shows the possible indications of the LED and the corresponding binary values of the active segments in order to have a hexadecimal indicator. Note that a segment is turned ON when the corresponding GPIO pin becomes LOW. The OFF state will be when all segments are "1".

Indication	Binary value [a b c d e f g dp]
0	00000011
1	10011111
2	00100101
3	00001101
4	10011001
5	01001001
6	11000001
7	00011111
8	00000001
9	00001001
A	00010001
b	11000001
C	01100011
d	10000101
E	01100001
F	01110001

Example-1

Make a program that makes the 7-segment LED a hexadecimal counter.

```
1  import RPi.GPIO as GPIO
2  from time import sleep
3
4  #pin definitions
5  seg_a=40
6  seg_b=31
7  seg_c=32
8  seg_d=33
9  seg_e=35
10 seg_f=36
11 seg_g=37
12 seg_dp=38
13
14 GPIO.setmode(GPIO.BOARD)
15
16 #segment pins
17 segments =  (seg_a, seg_b, seg_c, seg_d, seg_e,
               seg_f, seg_g, seg_dp)
18 #segment pins mode and initial state
19 for segment in segments:
20    GPIO.setup(segment, GPIO.OUT)
21    GPIO.output(segment, GPIO.HIGH)
22 #bit patterns
23 num = [[1,1,1,1,1,1,1,1], #OFF
24        [0,0,0,0,0,0,1,1], #'0'
25        [1,0,0,1,1,1,1,1], #'1'
26        [0,0,1,0,0,1,0,1], #'2'
27        [0,0,0,0,1,1,0,1], #'3'
28        [1,0,0,1,1,0,0,1], #'4'
29        [0,1,0,0,1,0,0,1], #'5'
30        [1,1,0,0,0,0,0,1], #'6'
31        [0,0,0,1,1,1,1,1], #'7'
32        [0,0,0,0,0,0,0,1], #'8'
33        [0,0,0,0,1,0,0,1], #'9'
34        [0,0,0,1,0,0,0,1], #'A'
35        [1,1,0,0,0,0,0,1], #'b'
36        [0,1,1,0,0,0,1,1], #'C'
37        [1,0,0,0,0,1,0,1], #'d'
38        [0,1,1,0,0,0,0,1], #'E'
39        [0,1,1,1,0,0,0,1]] #'F'
```

```
40 try:
41    while True:
42        for i in range(0, 17):
43            for j in range (0, 7):
44                GPIO.output(segments[j], num[i][j])
45            sleep(2)
46
47 except KeyboardInterrupt:
48    GPIO.cleanup()
```

- Lines 5 to 12 define the pin connections according to Fig 6.
- Line 14 chooses the BOARD numbering scheme.
- Lines 17 to 21 assign all pins as outputs with initial state HIGH.
- Lines 23 to 39 give a list with the bit patterns of the possible hexadecimal numbers as denoted in the above mentioned table.
- Lines 40-41 and 47 define a loop that can only be interrupted with CTRL+C.
- Lines 42 to 45 output the bit patterns to the predefined segment pins in order to show the hexadecimal digit for 2 seconds.
- Line 48 make all pins LOW.

Example-2

Make a program that in every press of Traffic Hat button the 7-segment LED will count from 0 to F.

```
1  import RPi.GPIO as GPIO
2  from time import sleep
3
4  #pin definitions
5  seg_a=40
6  seg_b=31
7  seg_c=32
8  seg_d=33
9  seg_e=35
10 seg_f=36
11 seg_g=37
12 seg_dp=38
13
14 btn=22
15
16 GPIO.setmode(GPIO.BOARD)
17
18 #Button has an external pull-up resistor
19 #GPIO.setup(btnPin, GPIO.IN)
20 #Button has no external pull-up resistor
21 GPIO.setup(btn, GPIO.IN, pull_up_down=GPIO.PUD_UP)
22 #segment pins
23 segments =  (seg_a, seg_b, seg_c, seg_d, seg_e,
               seg_f, seg_g, seg_dp)
24 #segment pins mode and initial state
25 for segment in segments:
26    GPIO.setup(segment, GPIO.OUT)
27    GPIO.output(segment, GPIO.HIGH)
28
29 #bit patterns
30 num = [[1,1,1,1,1,1,1,1],#OFF
31       [0,0,0,0,0,0,1,1], #'0'
32       [1,0,0,1,1,1,1,1], #'1'
33       [0,0,1,0,0,1,0,1], #'2'
34       [0,0,0,0,1,1,0,1], #'3'
35       [1,0,0,1,1,0,0,1], #'4'
36       [0,1,0,0,1,0,0,1], #'5'
37       [1,1,0,0,0,0,0,1], #'6'
38       [0,0,0,1,1,1,1,1], #'7'
39       [0,0,0,0,0,0,0,1], #'8'
40       [0,0,0,0,1,0,0,1], #'9'
41       [0,0,0,1,0,0,0,1], #'A'
42       [1,1,0,0,0,0,0,1], #'b'
43       [0,1,1,0,0,0,1,1], #'C'
44       [1,0,0,0,0,1,0,1], #'d'
45       [0,1,1,0,0,0,0,1], #'E'
46       [0,1,1,1,0,0,0,1]] #'F'
```

```
47 try:
48     while True:
49         if not GPIO.input(btn):
50             while not GPIO.input(btn):
51                 for i in range(0, 17):
52                     for j in range (0, 7):
53                         GPIO.output(segments[j], num[i][j])
54                     sleep(1)
55
56 except KeyboardInterrupt:
57     GPIO.cleanup()
```

- Lines 5 to 12 define the pin connections according to Fig 6.
- Line 14 defines the pin for the button.
- Line 16 chooses the BOARD numbering scheme.
- Line 21 makes the pin for the button as input with internal pull-up resistor.
- Lines 23 to 27 assign all pins as outputs with initial state HIGH.
- Lines 30 to 46 give a list with the bit patterns of the possible hexadecimal numbers as denoted in the above mentioned table.
- Lines 47-48 and 56 define a loop that can only be interrupted with CTRL+C.
- Lines 49-50 detect a full press of the button (HIGH-LOW-HIGH)
- Lines 51 to 54 output the bit patterns to the predefined segment pins in order to show the hexadecimal digit for 1 second.
- Line 57 make all pins LOW.

Drive a dot matrix LED

EXcept from the 7-segment LEDs that can show a limited number of characters, the dot-matrix LEDs are very popular for displaying and scrolling text messages. Fig 6 shows such a module which is a serially driven 8x8 LED Matrix powered by MAX7219. The 8x8 LED Matrix is easy to use, because only needs three data lines and two power lines.

MAX7219 is an integrated serial input/output common-cathode display driver that can be connected to a microcontroller in order to drive an 8-digit 7-segment digital LED display or a bar graph display or 64 separate LEDs. It is worth mentioning that the module can be cascaded in order to create a long display (up to 8 modules).

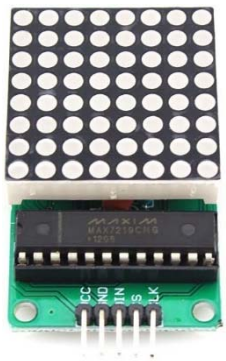


Fig 6: 8X8 dot-matrix LED.

Fig 7 shows the diodes behind each dot and their connection to digit or segment pins of the module.

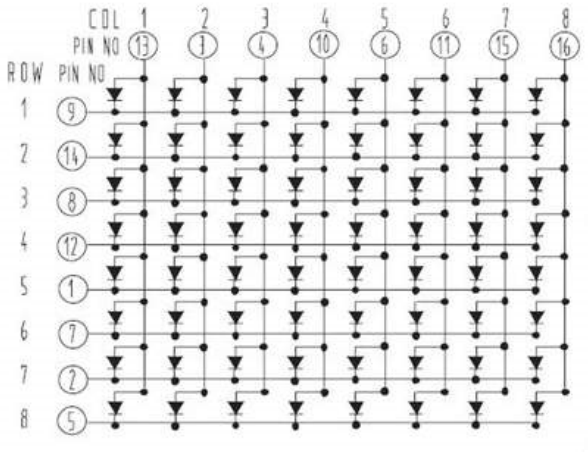
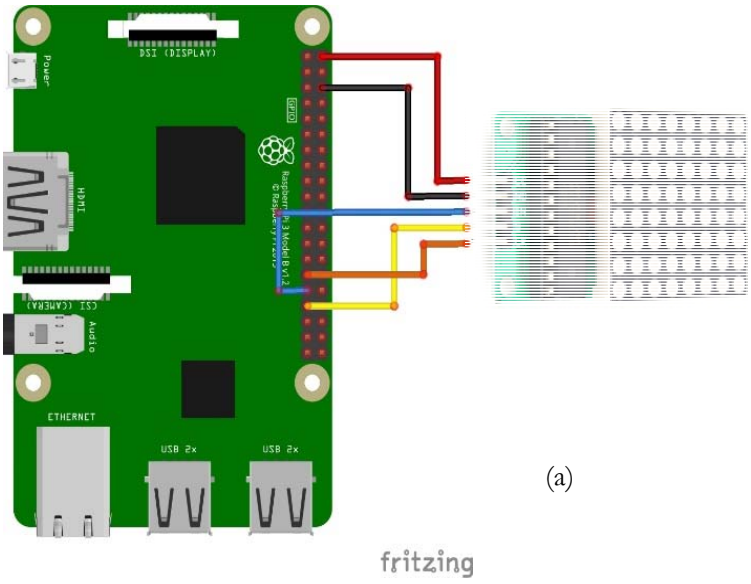
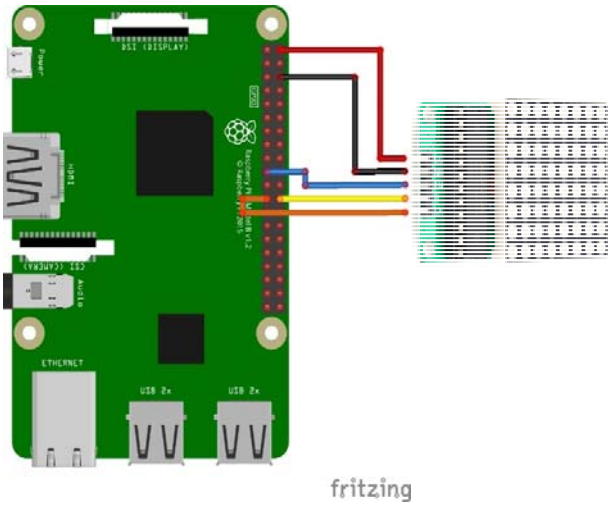


Fig 7: LED diodes of the dot-matrix.

This module can be driven by raspberry pi either by using general GPIO pins, simulating the timing diagrams of MAX7219 or by using SPI interface. Fig 8 shows the connection of the dot matrix with raspberry pi with these two ways of driving.



(a)



(b)

Fig 8: (a) 8x8 LED matrix is connected to Raspberry pi with GPIO pins.
(b) 8x8 LED matrix is connected to Raspberry pi with SPI pins.

The following table has the pin connections with the two ways of driving.

GPIO PINS	8x8 MATRIX	SPI PINS
2 (+5V)	**Vcc**	2 (+5V)
6 (GND)	**GND**	6 (GND)
31 (GPIO06)	**DIN**	19 (GPIO10)
33 (GPIO13)	**CS**	24 (GPIO08)
29 (GPIO05)	**CLK**	23 (GPIO11)

In that sense, three coding examples follow covering the two ways of driving without using an existing library. The first example shows a message by using the general GPIO pins. The second example shows a message by using the SPI interface of Raspberry pi. The third example shows a scrolling message by using the SPI interface on a single matrix module. Finally, the fourth example presents a scrolling message on two interconnected modules using general GPIO pins.

Example-1

Make a program that drives an 8x8 dot-matrix LED module with MAX7219 and shows a text message. Driving is carried out with general purpose GPIO pins, simulating the timing diagram of MAX7219.

```
#8X8 LED MATRIX WITHOUT USING SPI
import RPi.GPIO as GPIO
from time import sleep
from signal import pause

pinCLK = 29
pinDIN = 31
pinCE = 33

#MAX7219 registers
MAX7219_NOOP = 0x00
MAX7219_DIGIT0 = 0x01
MAX7219_DIGIT1 = 0x02
MAX7219_DIGIT2 = 0x03
MAX7219_DIGIT3 = 0x04
MAX7219_DIGIT4 = 0x05
MAX7219_DIGIT5 = 0x06
MAX7219_DIGIT6 = 0x07
MAX7219_DIGIT7 = 0x08
MAX7219_DECODEMODE = 0x09
MAX7219_INTENSITY = 0x0A
MAX7219_SCANLIMIT = 0x0B
MAX7219_SHUTDOWN = 0x0C
MAX7219_TEST = 0x0F

text='THIS EXAMPLE IS ABOUT DRIVING A LED MATRIX WITH
      MAX7219 WITHOUT SPI'

cp437_font = {'0':( 0x3E, 0x7F, 0x71, 0x59, 0x4D, 0x7F,
                    0x3E, 0x00 ), # '0'
              '1':( 0x40, 0x42, 0x7F, 0x7F, 0x40, 0x40,
                    0x00, 0x00 ), # '1'
              '2':( 0x62, 0x73, 0x59, 0x49, 0x6F, 0x66,
                    0x00, 0x00 ), # '2'
              '3':( 0x22, 0x63, 0x49, 0x49, 0x7F, 0x36,
                    0x00, 0x00 ), # '3'
              '4':( 0x18, 0x1C, 0x16, 0x53, 0x7F, 0x7F,
                    0x50, 0x00 ), # '4'
              '5':( 0x27, 0x67, 0x45, 0x45, 0x7D, 0x39,
                    0x00, 0x00 ), # '5'
              '6':( 0x3C, 0x7E, 0x4B, 0x49, 0x79, 0x30,
                    0x00, 0x00  ), # '6'
              '7':( 0x03, 0x03, 0x71, 0x79, 0x0F, 0x07,
                    0x00, 0x00 ), # '7'
              '8':( 0x36, 0x7F, 0x49, 0x49, 0x7F, 0x36,
                    0x00, 0x00 ), # '8'
              '9':( 0x06, 0x4F, 0x49, 0x69, 0x3F, 0x1E,
                    0x00, 0x00 ), # '9'
              ' ':( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                    0x00, 0x00 ), # ' '
```

```
39              'A':( 0x7C, 0x7E, 0x13, 0x13, 0x7E, 0x7C,
                      0x00, 0x00 ), # 'A'
40              'B':( 0x41, 0x7F, 0x7F, 0x49, 0x49, 0x7F,
                      0x36, 0x00 ), # 'B'
41              'C':( 0x1C, 0x3E, 0x63, 0x41, 0x41, 0x63,
                      0x22, 0x00 ), # 'C'
42              'D':( 0x41, 0x7F, 0x7F, 0x41, 0x63, 0x3E,
                      0x1C, 0x00 ), # 'D'
43              'E':( 0x41, 0x7F, 0x7F, 0x49, 0x5D, 0x41,
                      0x63, 0x00 ), # 'E'
44              'F':( 0x41, 0x7F, 0x7F, 0x49, 0x1D, 0x01,
                      0x03, 0x00 ), # 'F'
45              'G':( 0x1C, 0x3E, 0x63, 0x41, 0x51, 0x73,
                      0x72, 0x00 ), # 'G'
46              'H':( 0x7F, 0x7F, 0x08, 0x08, 0x7F, 0x7F,
                      0x00, 0x00 ), # 'H'
47              'I':( 0x00, 0x41, 0x7F, 0x7F, 0x41, 0x00,
                      0x00, 0x00 ), # 'I'
48              'J':( 0x30, 0x70, 0x40, 0x41, 0x7F, 0x3F,
                      0x01, 0x00 ), # 'J'
49              'K':( 0x41, 0x7F, 0x7F, 0x08, 0x1C, 0x77,
                      0x63, 0x00 ), # 'K'
50              'L':( 0x41, 0x7F, 0x7F, 0x41, 0x40, 0x60,
                      0x70, 0x00 ), # 'L'
51              'M':( 0x7F, 0x7F, 0x0E, 0x1C, 0x0E, 0x7F,
                      0x7F, 0x00 ), # 'M'
52              'N':( 0x7F, 0x7F, 0x06, 0x0C, 0x18, 0x7F,
                      0x7F, 0x00 ), # 'N'
53              'O':( 0x1C, 0x3E, 0x63, 0x41, 0x63, 0x3E,
                      0x1C, 0x00 ), # 'O'
54              'P':( 0x41, 0x7F, 0x7F, 0x49, 0x09, 0x0F,
                      0x06, 0x00 ), # 'P'
55              'Q':( 0x1E, 0x3F, 0x21, 0x71, 0x7F, 0x5E,
                      0x00, 0x00 ), # 'Q'
56              'R':( 0x41, 0x7F, 0x7F, 0x09, 0x19, 0x7F,
                      0x66, 0x00 ), # 'R'
57              'S':( 0x26, 0x6F, 0x4D, 0x59, 0x73, 0x32,
                      0x00, 0x00 ), # 'S'
58              'T':( 0x03, 0x41, 0x7F, 0x7F, 0x41, 0x03,
                      0x00, 0x00 ), # 'T'
59              'U':( 0x7F, 0x7F, 0x40, 0x40, 0x7F, 0x7F,
                      0x00, 0x00 ), # 'U'
60              'V':( 0x1F, 0x3F, 0x60, 0x60, 0x3F, 0x1F,
                      0x00, 0x00 ), # 'V'
61              'W':( 0x7F, 0x7F, 0x30, 0x18, 0x30, 0x7F,
                      0x7F, 0x00 ), # 'W'
62              'X':( 0x43, 0x67, 0x3C, 0x18, 0x3C, 0x67,
                      0x43, 0x00 ), # 'X'
63              'Y':( 0x07, 0x4F, 0x78, 0x78, 0x4F, 0x07,
                      0x00, 0x00 ), # 'Y'
64              'Z':( 0x47, 0x63, 0x71, 0x59, 0x4D, 0x67,
                      0x73, 0x00 ), # 'Z'
65              }
```

```
66
67 GPIO.setmode(GPIO.BOARD)
68 GPIO.setup(pinCLK, GPIO.OUT)
69 GPIO.setup(pinCE, GPIO.OUT)
70 GPIO.setup(pinDIN, GPIO.OUT)
71 GPIO.output(pinCE, GPIO.HIGH)
72 GPIO.output(pinCLK, GPIO.HIGH)
73
74 def write_MAX_byte(data):
75     for i in range(0,8):
76         GPIO.output(pinCLK, GPIO.LOW)
77         GPIO.output(pinDIN, data & 0x80)
78         data=data<<1
79         GPIO.output(pinCLK, GPIO.HIGH)
80
81 def write_MAX(max_address, max_data):
82     GPIO.output(pinCE, GPIO.LOW)
83     write_MAX_byte(max_address)          #column 0-7
84     write_MAX_byte(max_data)             #binary pattern
85     GPIO.output(pinCE, GPIO.HIGH)
86
87 def init_MAX():
88     write_MAX(MAX7219_DECODEMODE, 0x00)
89     write_MAX(MAX7219_INTENSITY, 0x03)
90     write_MAX(MAX7219_SCANLIMIT, 0x07)
91     write_MAX(MAX7219_SHUTDOWN, 0x01)
92     write_MAX(MAX7219_TEST, 0x00)
93
94 init_MAX()
95 sleep(0.5)
96
97 for i in range(len(text)):
98     for j in range(0, 8):
99         write_MAX(j+1, cp437_font[text[i]][j])
100    sleep(1)
101
102 write_MAX(MAX7219_SHUTDOWN, 0x00)
103 pause()
```

- Lines 6 to 8 define the GPIO pins that are connected to MAX7219 (CLK, DIN and CE).
- Lines 11 to 24 define the internal register addresses of MAX7219 that are for the digits and its operating modes.
- Line 26 is the text message that will be indicated on the single 8X8 LED matrix.

- Lines 28 to 65 define the 8 bytes needed in order to show a character on the matrix. The characters used are the numbers, the capital letters and the space. The user may enter any character in the same manner.
- Line 67 chooses the BOARD numbering scheme.
- Lines 68 to 72 define the communication pins as outputs and their initial state.
- Lines 74 to 79 define the function *write_MAX_byte()* which outputs a byte to MAX7219. Raspberry outputs the most significant bit of the byte in every CLK rising edge and shifts the byte to the left eight times.
- Lines 81 to 85 define the function *write_MAX()* which outputs firstly the address and then the data to MAX7219 using the previous method.
- Lines 87 to 92 define the function *init_MAX()* which initializes the MAX7219 with the proper operating modes.
- Line 94 initializes MAX7219 through the previous method.
- Line 95 gives a small delay for setup.
- Line 97 scans the characters of the text.
- Lines 98-99 write the 8 bytes that correspond to the character of the text.
- Line 100 makes a delay of 1 second before next character is displayed.
- Line 102 erases the display after all text is shown.
- Line 101 pauses the program execution and user may exit with CTRL+C.

Example-2

Make a program that drives an 8x8 dot-matrix LED module with MAX7219 and shows a text message. Driving is carried out with SPI pins using library "spidev".

Do not forget to connect the MAX7219 pins to the SPI pins as indicated in the above table.

```
1 #8X8 LED MATRIX USING SPI
2 import spidev
3 import RPi.GPIO as GPIO
4 from time import sleep
5 from signal import pause
6
```

```
7  #MAX7219 registers
8  MAX7219_NOOP = 0x00
9  MAX7219_DIGIT0 = 0x01
10 MAX7219_DIGIT1 = 0x02
11 MAX7219_DIGIT2 = 0x03
12 MAX7219_DIGIT3 = 0x04
13 MAX7219_DIGIT4 = 0x05
14 MAX7219_DIGIT5 = 0x06
15 MAX7219_DIGIT6 = 0x07
16 MAX7219_DIGIT7 = 0x08
17 MAX7219_DECODEMODE = 0x09
18 MAX7219_INTENSITY = 0x0A
19 MAX7219_SCANLIMIT = 0x0B
20 MAX7219_SHUTDOWN = 0x0C
21 MAX7219_TEST = 0x0F
22
23 text='THIS EXAMPLE IS ABOUT DRIVING A LED MATRIX WITH
         MAX7219 WITH SPI'
24
25 cp437_font = {'0':( 0x3E, 0x7F, 0x71, 0x59, 0x4D, 0x7F,
                       0x3E, 0x00 ), # '0'
26               '1':( 0x40, 0x42, 0x7F, 0x7F, 0x40, 0x40,
                       0x00, 0x00 ), # '1'
27               '2':( 0x62, 0x73, 0x59, 0x49, 0x6F, 0x66,
                       0x00, 0x00 ), # '2'
28               '3':( 0x22, 0x63, 0x49, 0x49, 0x7F, 0x36,
                       0x00, 0x00 ), # '3'
29               '4':( 0x18, 0x1C, 0x16, 0x53, 0x7F, 0x7F,
                       0x50, 0x00 ), # '4'
30               '5':( 0x27, 0x67, 0x45, 0x45, 0x7D, 0x39,
                       0x00, 0x00 ), # '5'
31               '6':( 0x3C, 0x7E, 0x4B, 0x49, 0x79, 0x30,
                       0x00, 0x00 ), # '6'
32               '7':( 0x03, 0x03, 0x71, 0x79, 0x0F, 0x07,
                       0x00, 0x00 ), # '7'
33               '8':( 0x36, 0x7F, 0x49, 0x49, 0x7F, 0x36,
                       0x00, 0x00 ), # '8'
34               '9':( 0x06, 0x4F, 0x49, 0x69, 0x3F, 0x1E,
                       0x00, 0x00 ), # '9'
35               ' ':( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                       0x00, 0x00 ), # ' '
36               'A':( 0x7C, 0x7E, 0x13, 0x13, 0x7E, 0x7C,
                       0x00, 0x00 ), # 'A'
37               'B':( 0x41, 0x7F, 0x7F, 0x49, 0x49, 0x7F,
                       0x36, 0x00 ), # 'B'
38               'C':( 0x1C, 0x3E, 0x63, 0x41, 0x41, 0x63,
                       0x22, 0x00 ), # 'C'
39               'D':( 0x41, 0x7F, 0x7F, 0x41, 0x63, 0x3E,
                       0x1C, 0x00 ), # 'D'
```

```
40              'E':( 0x41, 0x7F, 0x7F, 0x49, 0x5D, 0x41,
                      0x63, 0x00 ), # 'E'
41              'F':( 0x41, 0x7F, 0x7F, 0x49, 0x1D, 0x01,
                      0x03, 0x00 ), # 'F'
42              'G':( 0x1C, 0x3E, 0x63, 0x41, 0x51, 0x73,
                      0x72, 0x00 ), # 'G'
43              'H':( 0x7F, 0x7F, 0x08, 0x08, 0x7F, 0x7F,
                      0x00, 0x00 ), # 'H'
44              'I':( 0x00, 0x41, 0x7F, 0x7F, 0x41, 0x00,
                      0x00, 0x00 ), # 'I'
45              'J':( 0x30, 0x70, 0x40, 0x41, 0x7F, 0x3F,
                      0x01, 0x00 ), # 'J'
46              'K':( 0x41, 0x7F, 0x7F, 0x08, 0x1C, 0x77,
                      0x63, 0x00 ), # 'K'
47              'L':( 0x41, 0x7F, 0x7F, 0x41, 0x40, 0x60,
                      0x70, 0x00 ), # 'L'
48              'M':( 0x7F, 0x7F, 0x0E, 0x1C, 0x0E, 0x7F,
                      0x7F, 0x00 ), # 'M'
49              'N':( 0x7F, 0x7F, 0x06, 0x0C, 0x18, 0x7F,
                      0x7F, 0x00 ), # 'N'
50              'O':( 0x1C, 0x3E, 0x63, 0x41, 0x63, 0x3E,
                      0x1C, 0x00 ), # 'O'
51              'P':( 0x41, 0x7F, 0x7F, 0x49, 0x09, 0x0F,
                      0x06, 0x00 ), # 'P'
52              'Q':( 0x1E, 0x3F, 0x21, 0x71, 0x7F, 0x5E,
                      0x00, 0x00 ), # 'Q'
53              'R':( 0x41, 0x7F, 0x7F, 0x09, 0x19, 0x7F,
                      0x66, 0x00 ), # 'R'
54              'S':( 0x26, 0x6F, 0x4D, 0x59, 0x73, 0x32,
                      0x00, 0x00 ), # 'S'
55              'T':( 0x03, 0x41, 0x7F, 0x7F, 0x41, 0x03,
                      0x00, 0x00 ), # 'T'
56              'U':( 0x7F, 0x7F, 0x40, 0x40, 0x7F, 0x7F,
                      0x00, 0x00 ), # 'U'
57              'V':( 0x1F, 0x3F, 0x60, 0x60, 0x3F, 0x1F,
                      0x00, 0x00 ), # 'V'
58              'W':( 0x7F, 0x7F, 0x30, 0x18, 0x30, 0x7F,
                      0x7F, 0x00 ), # 'W'
59              'X':( 0x43, 0x67, 0x3C, 0x18, 0x3C, 0x67,
                      0x43, 0x00 ), # 'X'
60              'Y':( 0x07, 0x4F, 0x78, 0x78, 0x4F, 0x07,
                      0x00, 0x00 ), # 'Y'
61              'Z':( 0x47, 0x63, 0x71, 0x59, 0x4D, 0x67,
                      0x73, 0x00 ), # 'Z'
62              }
63
64 def sendSPI(address, data):
65     spi.xfer([address, data]) #Send address&data to MAX7219
66
67 GPIO.setmode(GPIO.BOARD)
68 spi = spidev.SpiDev()          #create spi object
69 spi.open(0, 0)                 #open spi port 0, device (CE)0
```

```
70
71 #MAX7219 initialization
72 sendSPI(MAX7219_TEST, 0x00)         #Finish test
73 sendSPI(MAX7219_DECODEMODE, 0x00)   #Disable BCD mode
74 sendSPI(MAX7219_INTENSITY, 0x03)    #Lowest intensity
75 sendSPI(MAX7219_SCANLIMIT, 0x07)    #Scan all digits
76 sendSPI(MAX7219_SHUTDOWN, 0x01)     #Turn on chip
77
78 for i in range(len(text)):
79    for j in range(0, 8):
80        sendSPI(j+1, cp437_font[text[i]][j])
81    sleep(1)
82
83 sendSPI(MAX7219_SHUTDOWN, 0x00)     #Turn off chip
84 spi.close()
85 pause()
```

- Line 2 imports "spidev" library for SPI communication. Note that SPI is not activated when Raspberry pi is booted and therefore the user should activate it from the interfaces configuration panel.
- Note that there is no definition of SPI pins because they are unique.
- Lines 8 to 21 define the internal register addresses of MAX7219 that are for the digits and its operating modes.
- Line 23 is the text message that will be indicated on the single 8X8 LED matrix.
- Lines 25 to 62 define the 8 bytes needed in order to show a character on the matrix. The characters used are the numbers, the capital letters and the space.
- Lines 65-65 define the function *sendSPI()* which transfers the address and the corresponding data to MAX7219 with the SPI protocol using the *spi.xfer()* method.
- Line 67 chooses the BOARD numbering scheme.
- Line 68 instantiates an object of the SPI.
- Line 69 opens the port "0" with CE0. Note that there is also a CE1 pin.
- Lines 72 to 76 initialize MAX7219 to the proper operating modes.
- Line 78 scans the characters of the text.
- Lines 79-80 write the 8 bytes that correspond to the character of the text through the *sendSPI()* method.
- Line 81 makes a delay of 1 second between the displayed characters.

- Line 83 turns off MAX7219.
- Line 84 closes SPI channel.
- Line 85 pauses the program.

Example-3

Make a program that drives **one** 8x8 dot-matrix LED module with MAX7219 and shows a scrolling text message. Driving is carried out with SPI pins using library "spidev".

```
1  #8X8 LED MATRIX USING SPI - TEXT SCROLLING
2  import spidev
3  import RPi.GPIO as GPIO
4  from time import sleep
5  from signal import pause
6
7  #MAX7219 registers
8  MAX7219_NOOP = 0x00
9  MAX7219_DIGIT0 = 0x01
10 MAX7219_DIGIT1 = 0x02
11 MAX7219_DIGIT2 = 0x03
12 MAX7219_DIGIT3 = 0x04
13 MAX7219_DIGIT4 = 0x05
14 MAX7219_DIGIT5 = 0x06
15 MAX7219_DIGIT6 = 0x07
16 MAX7219_DIGIT7 = 0x08
17 MAX7219_DECODEMODE = 0x09
18 MAX7219_INTENSITY = 0x0A
19 MAX7219_SCANLIMIT = 0x0B
20 MAX7219_SHUTDOWN = 0x0C
21 MAX7219_TEST = 0x0F
22
23 text='TEXT SCROLL WITH MAX7219 AND SPI'
24
25 charWidth = 7
26 leftStart = -1
27 leftEnd = 8
28
29 cp437_font = {'0':( 0x3E, 0x7F, 0x71, 0x59, 0x4D, 0x7F,
       0x3E, 0x00 ),
30 '1':( 0x40, 0x42, 0x7F, 0x7F, 0x40, 0x40, 0x00, 0x00 ),
31 '2':( 0x62, 0x73, 0x59, 0x49, 0x6F, 0x66, 0x00, 0x00 ),
32 '3':( 0x22, 0x63, 0x49, 0x49, 0x7F, 0x36, 0x00, 0x00 ),
```

```
33      '4':( 0x18, 0x1C, 0x16, 0x53, 0x7F, 0x7F, 0x50, 0x00 ),
34      '5':( 0x27, 0x67, 0x45, 0x45, 0x7D, 0x39, 0x00, 0x00 ),
35      '6':( 0x3C, 0x7E, 0x4B, 0x49, 0x79, 0x30, 0x00, 0x00 ),
36      '7':( 0x03, 0x03, 0x71, 0x79, 0x0F, 0x07, 0x00, 0x00 ),
37      '8':( 0x36, 0x7F, 0x49, 0x49, 0x7F, 0x36, 0x00, 0x00 ),
38      '9':( 0x06, 0x4F, 0x49, 0x69, 0x3F, 0x1E, 0x00, 0x00 ),
39      ' ':( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 ),
40      'A':( 0x7C, 0x7E, 0x13, 0x13, 0x7E, 0x7C, 0x00, 0x00 ),
41      'B':( 0x41, 0x7F, 0x7F, 0x49, 0x49, 0x7F, 0x36, 0x00 ),
42      'C':( 0x1C, 0x3E, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00 ),
43      'D':( 0x41, 0x7F, 0x7F, 0x41, 0x63, 0x3E, 0x1C, 0x00 ),
44      'E':( 0x41, 0x7F, 0x7F, 0x49, 0x5D, 0x41, 0x63, 0x00 ),
45      'F':( 0x41, 0x7F, 0x7F, 0x49, 0x1D, 0x01, 0x03, 0x00 ),
46      'G':( 0x1C, 0x3E, 0x63, 0x41, 0x51, 0x73, 0x72, 0x00 ),
47      'H':( 0x7F, 0x7F, 0x08, 0x08, 0x7F, 0x7F, 0x00, 0x00 ),
48      'I':( 0x00, 0x41, 0x7F, 0x7F, 0x41, 0x00, 0x00, 0x00 ),
49      'J':( 0x30, 0x70, 0x40, 0x41, 0x7F, 0x3F, 0x01, 0x00 ),
50      'K':( 0x41, 0x7F, 0x7F, 0x08, 0x1C, 0x77, 0x63, 0x00 ),
51      'L':( 0x41, 0x7F, 0x7F, 0x41, 0x40, 0x60, 0x70, 0x00 ),
52      'M':( 0x7F, 0x7F, 0x0E, 0x1C, 0x0E, 0x7F, 0x7F, 0x00 ),
53      'N':( 0x7F, 0x7F, 0x06, 0x0C, 0x18, 0x7F, 0x7F, 0x00 ),
54      'O':( 0x1C, 0x3E, 0x63, 0x41, 0x63, 0x3E, 0x1C, 0x00 ),
55      'P':( 0x41, 0x7F, 0x7F, 0x49, 0x09, 0x0F, 0x06, 0x00 ),
56      'Q':( 0x1E, 0x3F, 0x21, 0x71, 0x7F, 0x5E, 0x00, 0x00 ),
57      'R':( 0x41, 0x7F, 0x7F, 0x09, 0x19, 0x7F, 0x66, 0x00 ),
58      'S':( 0x26, 0x6F, 0x4D, 0x59, 0x73, 0x32, 0x00, 0x00 ),
59      'T':( 0x03, 0x41, 0x7F, 0x7F, 0x41, 0x03, 0x00, 0x00 ),
60      'U':( 0x7F, 0x7F, 0x40, 0x40, 0x7F, 0x7F, 0x00, 0x00 ),
61      'V':( 0x1F, 0x3F, 0x60, 0x60, 0x3F, 0x1F, 0x00, 0x00 ),
62      'W':( 0x7F, 0x7F, 0x30, 0x18, 0x30, 0x7F, 0x7F, 0x00 ),
63      'X':( 0x43, 0x67, 0x3C, 0x18, 0x3C, 0x67, 0x43, 0x00 ),
64      'Y':( 0x07, 0x4F, 0x78, 0x78, 0x4F, 0x07, 0x00, 0x00 ),
65      'Z':( 0x47, 0x63, 0x71, 0x59, 0x4D, 0x67, 0x73, 0x00 ),
66      }
67
68  def sendSPI(address, value):
69      spi.xfer([address, data]) #Send address&data to MAX7219
70
71  GPIO.setmode(GPIO.BOARD)
72  spi = spidev.SpiDev()       # create spi object
73  spi.open(0, 0)              # open spi port 0, device (CE) 0
74
```

```
75 #MAX7219 initialization
76 sendSPI(MAX7219_TEST, 0x00)         # Finish test
77 sendSPI(MAX7219_DECODEMODE, 0x00)   # Disable BCD mode
78 sendSPI(MAX7219_INTENSITY, 0x03)    # Lowest intensity
79 sendSPI(MAX7219_SCANLIMIT, 0x07)    # Scan all digits
80 sendSPI(MAX7219_SHUTDOWN, 0x01)     # Turn on chip
81
82 while True:
83     cols=[0]*8
84     for i in range(len(text)):
85         for col in range(0,8):
86             pos = i * charWidth+col+leftStart+leftEnd
87             if pos >= 0 and pos < 8:
88                 cols[pos] = cp437_font[text[i]][col]
89     for col in range(0, 8):
90         sendSPI(col % 8+1, cols[col])
91
92     sleep(0.5)
93     leftStart=leftStart-1;
94     if leftStart==-8*len(text):
95         leftStart=-1;
```

- Line 2 imports "spidev" library for SPI communication. Note that SPI is not activated when Raspberry pi is booted and therefore the user should activate it from the interfaces configuration panel.
- Note that there is no definition of SPI pins because they are unique.
- Lines 8 to 21 define the internal register addresses of MAX7219 that are for the digits and its operating modes.
- Line 23 is the text message that will scroll on the single 8X8 LED matrix.
- Line 25 defines the character width which is 8 bytes (counting from 0 to 7).
- Line 26 defines the starting point of the displayed digits.
- Line 27 defines the end point of the displayed digits.
- Lines 29 to 66 define the 8 bytes needed in order to show a character on the matrix (character font). The characters used are the numbers, the capital letters and the space.
- Lines 68 and 69 define the function *sendSPI()* which sends to the matrix device an address and its corresponding data through SPI.
- Line 71 sets GPIO to BOARD mode.
- Line 72 instantiates an object of the SPI.

- Line 73 opens the port "0" with CE0. Note that there is also a CE1 pin.
- Lines 75 to 80 initialize matrix device with the proper operating modes.
- Line 82 makes an infinite loop.
- Line 83 initializes a memory buffer of 8 bytes size.
- Lines 84 to 88 store to the "cols" buffer the columns of the character fonts for each text character. At the beginning (i=0 and col=0), in cols[7] is stored the first column of the font corresponding to the first text character. cols[0] to cols[6] remain zero and therefore the display will show only the first column of character "T".
- Lines 89 and 90 send the address and the data values to MAX7219 through SPI interface. "col % 8+1" gives the range "1 to 8" which is the address of the dot matrix device for every value of "col".
- Line 92 makes a small delay.
- Line 93 reduces leftStart variable by one. This makes the first column of the first character font to move one place to the left. So, the display will show the first two columns, then the first three columns and so on.
- Lines 94 and 95 restore leftStart value to -1 if its value exceeds the number of bytes of the text fonts. This makes the text to scroll to left one position each time until text finishes.

Example-4

Make a program that drives **two** 8x8 dot-matrix LED modules with MAX7219 and shows a scrolling text message. Driving is carried out using GPIO pins for controlling the dot matrix device.

The setup for this example is shown in Fig. 9.

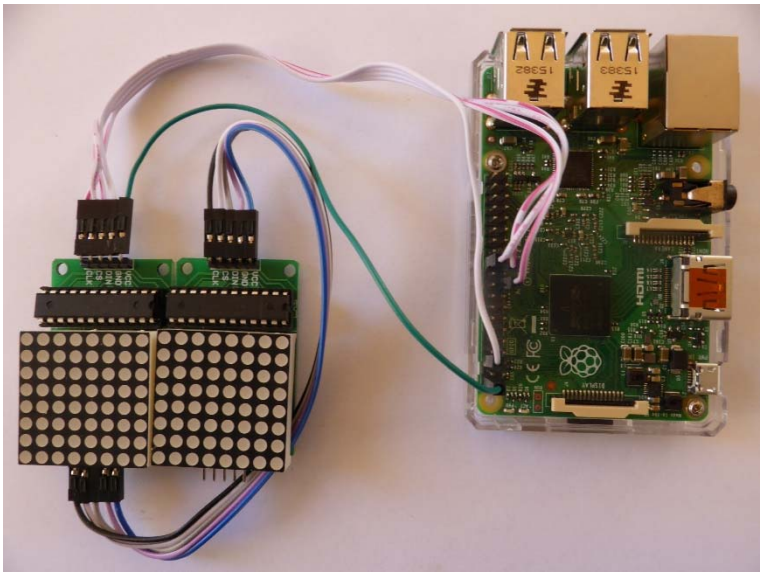


Fig 9: Scrolling text over 2 8x8 dot-matrix LED modules.

```
#8X8 LED MATRIX - TEXT SCROLLING
import RPi.GPIO as GPIO
from time import sleep
from signal import pause

pinDIN=19
pinCLK=23
pinCE=24

#MAX7219 registers
MAX7219_NOOP = 0x00
MAX7219_DIGIT0 = 0x01
MAX7219_DIGIT1 = 0x02
MAX7219_DIGIT2 = 0x03
MAX7219_DIGIT3 = 0x04
MAX7219_DIGIT4 = 0x05
MAX7219_DIGIT5 = 0x06
MAX7219_DIGIT6 = 0x07
MAX7219_DIGIT7 = 0x08
MAX7219_DECODEMODE = 0x09
MAX7219_INTENSITY = 0x0A
MAX7219_SCANLIMIT = 0x0B
MAX7219_SHUTDOWN = 0x0C
MAX7219_TEST = 0x0F

```

```
26 text='THIS EXAMPLE IS ABOUT TEXT SCROLLING WITH MAX7219'
27
28 charWidth = 7
29 numberOfDevices = 2
30 mySize=numberOfDevices*8
31 leftStart=-1
32 leftEnd=numberOfDevices*8
33
34 cp437_font = {  '0':( 0x3E, 0x7F, 0x71, 0x59, 0x4D, 0x7F,
       0x3E, 0x00 ),
35    '1':( 0x40, 0x42, 0x7F, 0x7F, 0x40, 0x40, 0x00, 0x00 ),
36    '2':( 0x62, 0x73, 0x59, 0x49, 0x6F, 0x66, 0x00, 0x00 ),
37    '3':( 0x22, 0x63, 0x49, 0x49, 0x7F, 0x36, 0x00, 0x00 ),
38    '4':( 0x18, 0x1C, 0x16, 0x53, 0x7F, 0x7F, 0x50, 0x00 ),
39    '5':( 0x27, 0x67, 0x45, 0x45, 0x7D, 0x39, 0x00, 0x00 ),
40    '6':( 0x3C, 0x7E, 0x4B, 0x49, 0x79, 0x30, 0x00, 0x00 ),
41    '7':( 0x03, 0x03, 0x71, 0x79, 0x0F, 0x07, 0x00, 0x00 ),
42    '8':( 0x36, 0x7F, 0x49, 0x49, 0x7F, 0x36, 0x00, 0x00 ),
43    '9':( 0x06, 0x4F, 0x49, 0x69, 0x3F, 0x1E, 0x00, 0x00 ),
44    ' ':( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 ),
45    'A':( 0x7C, 0x7E, 0x13, 0x13, 0x7E, 0x7C, 0x00, 0x00 ),
46    'B':( 0x41, 0x7F, 0x7F, 0x49, 0x49, 0x7F, 0x36, 0x00 ),
47    'C':( 0x1C, 0x3E, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00 ),
48    'D':( 0x41, 0x7F, 0x7F, 0x41, 0x63, 0x3E, 0x1C, 0x00 ),
49    'E':( 0x41, 0x7F, 0x7F, 0x49, 0x5D, 0x41, 0x63, 0x00 ),
50    'F':( 0x41, 0x7F, 0x7F, 0x49, 0x1D, 0x01, 0x03, 0x00 ),
51    'G':( 0x1C, 0x3E, 0x63, 0x41, 0x51, 0x73, 0x72, 0x00 ),
52    'H':( 0x7F, 0x7F, 0x08, 0x08, 0x7F, 0x7F, 0x00, 0x00 ),
53    'I':( 0x00, 0x41, 0x7F, 0x7F, 0x41, 0x00, 0x00, 0x00 ),
54    'J':( 0x30, 0x70, 0x40, 0x41, 0x7F, 0x3F, 0x01, 0x00 ),
55    'K':( 0x41, 0x7F, 0x7F, 0x08, 0x1C, 0x77, 0x63, 0x00 ),
56    'L':( 0x41, 0x7F, 0x7F, 0x41, 0x40, 0x60, 0x70, 0x00 ),
57    'M':( 0x7F, 0x7F, 0x0E, 0x1C, 0x0E, 0x7F, 0x7F, 0x00 ),
58    'N':( 0x7F, 0x7F, 0x06, 0x0C, 0x18, 0x7F, 0x7F, 0x00 ),
59    'O':( 0x1C, 0x3E, 0x63, 0x41, 0x63, 0x3E, 0x1C, 0x00 ),
60    'P':( 0x41, 0x7F, 0x7F, 0x49, 0x09, 0x0F, 0x06, 0x00 ),
61    'Q':( 0x1E, 0x3F, 0x21, 0x71, 0x7F, 0x5E, 0x00, 0x00 ),
62    'R':( 0x41, 0x7F, 0x7F, 0x09, 0x19, 0x7F, 0x66, 0x00 ),
63    'S':( 0x26, 0x6F, 0x4D, 0x59, 0x73, 0x32, 0x00, 0x00 ),
64    'T':( 0x03, 0x41, 0x7F, 0x7F, 0x41, 0x03, 0x00, 0x00 ),
65    'U':( 0x7F, 0x7F, 0x40, 0x40, 0x7F, 0x7F, 0x00, 0x00 ),
66    'V':( 0x1F, 0x3F, 0x60, 0x60, 0x3F, 0x1F, 0x00, 0x00 ),
67    'W':( 0x7F, 0x7F, 0x30, 0x18, 0x30, 0x7F, 0x7F, 0x00 ),
68    'X':( 0x43, 0x67, 0x3C, 0x18, 0x3C, 0x67, 0x43, 0x00 ),
69    'Y':( 0x07, 0x4F, 0x78, 0x78, 0x4F, 0x07, 0x00, 0x00 ),
70    'Z':( 0x47, 0x63, 0x71, 0x59, 0x4D, 0x67, 0x73, 0x00 ),
71    }
72
73 GPIO.setmode(GPIO.BOARD)
74 GPIO.setup(pinCLK, GPIO.OUT)
75 GPIO.setup(pinCE, GPIO.OUT)
```

```
76 GPIO.setup(pinDIN, GPIO.OUT)
77 GPIO.output(pinCE, GPIO.HIGH)
78 GPIO.output(pinCLK, GPIO.HIGH)
79
80 def write_MAX_byte(data):
81     for i in range(0,8):
82         GPIO.output(pinCLK, GPIO.LOW)
83         GPIO.output(pinDIN, data & 0x80)
84         data=data<<1
85         GPIO.output(pinCLK, GPIO.HIGH)
86
87 def send_CMD_MAX(address, value):
88     GPIO.output(pinCE, GPIO.LOW)
89     for i in range(0, numberOfDevices):
90         write_MAX_byte(address)
91         write_MAX_byte(value)
92     GPIO.output(pinCE, GPIO.HIGH)
93
94 send_CMD_MAX(MAX7219_DECODEMODE, 0x00)
95 send_CMD_MAX(MAX7219_INTENSITY, 0x03)
96 send_CMD_MAX(MAX7219_SCANLIMIT, 0x07)
97 send_CMD_MAX(MAX7219_SHUTDOWN, 0x01)
98 send_CMD_MAX(MAX7219_TEST, 0x00)
99
100 sleep(0.5)
101
102 while True:
103     cols=[0]*mySize
104     for i in range(len(text)):
105         for col in range(0,8):
106             pos = i * charWidth+col+leftStart+leftEnd
107             if pos >= 0 and pos < numberOfDevices * 8:
108                 cols[pos] = cp437_font[text[i]][col]
109     for col in range(0, numberOfDevices*8):
110         GPIO.output(pinCE, GPIO.LOW)
111         for i in range(0, numberOfDevices):
112             if i==col/8:
113                write_MAX_byte(col%8+1)
114                write_MAX_byte(cols[col])
115             else:
116                write_MAX_byte(0)
117                write_MAX_byte(0)
118         GPIO.output(pinCE, GPIO.HIGH)
119     sleep(0.5)
120     leftStart=leftStart-1;
121     if leftStart==-8*len(text):
122         leftStart=-1;
```

- Line 2 imports GPIO library.
- Lines 6 to 8 define the control signals of MAX7219 (DIN, CLK, CE).
- Lines 11 to 24 define the internal register addresses of MAX7219 that are for the digits and its operating modes.
- Line 26 is the text message that will scroll on the two 8X8 LED matrix.
- Line 28 defines the character width which is 8 bytes (counting from 0 to 7).
- Line 29 defines the number of dot matrix devices.
- Line 30 defines the length of the buffer used for storing the fonts.
- Line 31 defines the starting point of the displayed digits.
- Line 32 defines the end point of the displayed digits.
- Lines 34 to 70 define the 8 bytes needed in order to show a character on the matrix (character font). The characters used are the numbers, the capital letters and the space.
- Line 73 sets GPIO to BOARD mode.
- Lines 74 to 78 program the controlling pins DIN, CLK and CE.
- Lines 80 to 85 write a byte to MAX7219 controlling CLK and DIN.
- Lines 87 to 92 write a command byte to a specific address of MAX7219 for all devices.
- Lines 94 to 98 initialize matrix devices with the proper operating modes.
- Line 102 makes an infinite loop.
- Line 103 initializes a memory buffer with size "number of devices" * 8 bytes.
- Lines 104 to 108 store to the "cols" buffer the columns of the character fonts for each text character. At the beginning (i=0 and col=0), in cols[7] is stored the first column of the font corresponding to the first text character. cols[0] to cols[6] remain zero and therefore the display will show only the first column of character "T".
- Lines 109 to 118 send address and data values to each MAX7219. For columns 0 to 7 one data matrix module is programmed whereas for columns 8 to 15 the other module is programmed. "col % 8+1" gives the range "1 to 8" which is the address of the dot matrix device for every value of "col".
- Line 119 makes a small delay.

- Line 120 reduces leftStart variable by one. This makes the first column of the first character font to move one place to the left. So, the display will show the first two columns, then the first three columns and so on.
- Lines 121 and 122 restore leftStart value to -1 if its value exceeds the number of bytes of the text fonts. This makes the text to scroll to left one position each time until text finishes.