Exploration projects useful url:

<https://www.youtube.com/watch?v=rr_QKSyJkgM> – This a video from the person who created the python library for the MAX7219 display driver that we used for our project. In the video he shows an example of how to hook up the MAX7219 and 8 digit display to the pi. He also explains how the MAX7219 works, and he also explains how to use the python library with a sample program while explaining what each library function does.

<http://www.bristolwatch.com/ele2/rpi_clock_max7219.htm> - This is the web site were we got the python library for the MAX7219 display driver that we used for our project. This is also an explanation on how to use the library along with an example.

<https://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf> This is the datasheet for the MAX7219 display driver that we used for our project. It has an example circuit which helped connect it to the pi. Also includes the address for changing the brightness

Library used:

The library we used was made by Lewis Loflin that is meant to be used with the MAX7219 connected to an 8 digit 7 segment display: and can be found here: <http://www.bristolwatch.com/ele2/rpi_clock_max7219.htm>

We used this Library because it is easy to use, small, and can be copied into programs mainly without the need to download any libraries on the pi.

Below is a table describing what each of its function do:

|  |  |
| --- | --- |
| Function name | Description |
| pulseCLK | Sends a pulse through the clock pin. |
| pulseCS | Sends a pulse through the chip select pin. |
| ssrOut | Sends an address to the MAX7219 |
| initMAX7219 | Initializes the 8 digit display to show all 0s. |
| writeMAX7219 | Takes in a digit and a positions, then prints the digit at that position on the display. |
| displayOff | Turns display off. |
| displayOn | Turns display on. |

During our two projects we created functions that would increase the functionality of this library, they are show below:

|  |  |
| --- | --- |
| Function name | Description |
| set\_brightness | Sets brightness of display based number give to it. The number must be in range of 0-15 with 0 being the dimmest and 15 being the brightest. |

**Code we created:**

def set\_brightness(bright\_num):

bright\_adds=[0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F]

if(bright\_num>0 and bright\_num<15):

ssrOut(0x0A)

ssrOut(bright\_adds[bright\_num])

pulseCS()

def display\_num(num):

num=int(num)

num\_str=str(num)

l=len(num\_str)

if l>8:

print("Number > 8 digits")

else:

y=l-1

for x in range(1, 9):

if y<0:

writeMAX7219(0, x)

else:

writeMAX7219(int(num\_str[y]),y)

y=y-1

**Main Code:**

import RPi.GPIO as GPIO

import time

LATCH = 37 # CS

CLK = 23

dataBit = 19 # DIN

BUTTON\_UP=32

BUTTON\_DOWN=36

SLEEP=0.1

START\_BRIGHT=8

MAX\_BRIGHT=15

MIN\_BRIGHT=0

GPIO.setmode(GPIO.BOARD)

GPIO.setup(LATCH, GPIO.OUT) # P0

GPIO.setup(CLK, GPIO.OUT) # P1

GPIO.setup(dataBit, GPIO.OUT) # P7

GPIO.setup(BUTTON\_UP,GPIO.IN)

GPIO.setup(BUTTON\_DOWN,GPIO.IN)

# Setup IO

GPIO.output(LATCH, 0)

GPIO.output(CLK, 0)

def pulseCLK():

GPIO.output(CLK, 1)

# time.sleep(.001)

GPIO.output(CLK, 0)

return

def pulseCS():

GPIO.output(LATCH, 1)

# time.sleep(.001)

GPIO.output(LATCH, 0)

return

# shift byte into MAX7219

# MSB out first!

def ssrOut(value):

for x in range(0,8):

temp = value & 0x80

if temp == 0x80:

GPIO.output(dataBit, 1) # data bit HIGH

else:

GPIO.output(dataBit, 0) # data bit LOW

pulseCLK()

value = value << 0x01 # shift left

return

# initialize MAX7219 4 digits BCD

def initMAX7219():

# set decode mode

ssrOut(0x09) # address

# ssrOut(0x00); // no decode

ssrOut(0xFF) # 4-bit BCD decode eight digits

pulseCS();

# set intensity

ssrOut(0x0A) # address

ssrOut(0x08) # 9/32

pulseCS()

# set scan limit 0-7

ssrOut(0x0B); # address

ssrOut(0x07) # 8 digits

# ssrOut(0x03) # 4 digits

pulseCS()

# set for normal operation

ssrOut(0x0C) # address

# ssrOut(0x00); // Off

ssrOut(0x01) # On

pulseCS()

# clear to all 0s.

for x in range(0,9):

ssrOut(x)

ssrOut(0)

pulseCS()

return

def writeMAX7219(data, location):

ssrOut(location)

ssrOut(data)

pulseCS()

return

def displayOff():

# set for normal operation

ssrOut(0x0C) # address

ssrOut(0x00); # Off

# ssrOut(0x01) # On

pulseCS()

def displayOn():

# set for normal operation

ssrOut(0x0C) # address

# ssrOut(0x00); # Off

ssrOut(0x01) # On

pulseCS()

def set\_brightness(bright\_num):

bright\_adds=[0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F]

if(bright\_num>MIN\_BRIGHT and bright\_num<MAX\_BRIGHT):

ssrOut(0x0A)

ssrOut(bright\_adds[bright\_num])

pulseCS()

initMAX7219()

# time returned as string

# But order is reversed relative to MAX7219.

str1 = time.strftime("%H:%M:%S")

# str2 = time.strftime("%d:%m:%Y")

print(str1)

# print len(str1) # 8

# x is digit position must be 1 to 8

# y is used for string pointer

try:

bright\_value=START\_BRIGHT

while 1:

y = 7

for x in range(1, 9):

if str1[y] == ":":

# output "-"

writeMAX7219(10, x)

y = y - 1

continue # back to if

# convert y char to integer

writeMAX7219(int(str1[y]), x)

y = y - 1

str1 = time.strftime("%H:%M:%S")

if(GPIO.input(BUTTON\_DOWN) ==True):

if bright\_value>MIN\_BRIGHT:

bright\_value-=1

if GPIO.input(BUTTON\_UP) == True:

if bright\_value<MAX\_BRIGHT:

bright\_value+=1

set\_brightness(bright\_value)

time.sleep(SLEEP)

except KeyboardInterrupt:

pass

print("Good by!")

time.sleep(0.5)

displayOff()

**Circuit Overview:**

The circuit has two buttons for input, the red button to increase brightness on the MAX7219 and the black one to decrease it. Each of these buttons are connected to their own GPIO pin on the Raspberry Pi. The display is also connected to the Pi at MOSI, a GPIO pin, SCLK, 5 V power, and a ground that is shared with the Pi and the buttons.

**Problems:**

We had some difficulty understanding why there were two sets of ssrOut functions called in the base code that we borrowed from online to initialize the display but we realized that the first ssrOut function is to “turn on” which part of the display needs to change and the second calling of the function is to actually change the setting. We found this out by taking a closer look at the datasheet and noticing that the address on the table above the intensity setting (0x0A) matched the intensity setting in the code we had.