SMART ALARM WITH RASPBERRY PI

Project Report

‘

UHasselt and KULeuven

Senne Colson

Brecht Achten

Contents

[Introduction 2](#_Toc27679871)

[Description 2](#_Toc27679872)

[Features 2](#_Toc27679873)

[Setup 2](#_Toc27679874)

[Code 4](#_Toc27679875)

[7-segment display 4](#_Toc27679876)

[Checking agenda and setting alarm 6](#_Toc27679877)

[Future work 8](#_Toc27679878)

[Conclusion 8](#_Toc27679879)

[Bibliography 9](#_Toc27679880)

# Introduction

This report describes the project called “Smart Alarm”. It was part of the courses Embedded Systems over a time period of 3 months.

The first chapter describes the project, followed by the features and functionalities. Next, the setup is described, and which components were used. In the next chapters the python code of the Raspberry Pi is explained alongside the implementation of the GPIO-pins. Finally, a conclusion was drawn.

# Description

Every evening, before you go to sleep, you have to check your agenda for the next day in order to set the right alarm. The next morning you are woken up, but probably can’t resist the temptation to snooze your alarm and get a few extra minutes of sleep. This routine does not just cause a lot of stress but can also make you known as a latecomer. Wouldn’t it be easy if there was a program that can check your agenda for you and set the right alarm based on your preferences? This project report will explain you how to implement these functionalities on a Raspberry Pi.

At the end of this project you will have learned:

* How to use the GPIO pins of a Raspberry Pi (for clock and camera)
* How to check your schedule in your agenda

# Features

1. When the program starts, the time will be displayed on a 7-segment display by default.
2. By default, you will be woken up at 8am.
3. However, whenever you have an appointment earlier than 9am, you will be woken up one hour in advance of your earliest appointment.
4. In order to turn off your alarm, you have to push a button.

# Setup

First, a 7-segment display was set up as shown below. During our course of Digital Electronics in our second year, we had to buy a HDSP-B09G. We chose to use this one, but any other 7-segment display will do as well. The HDSP-B09G has seven pins at the bottom and five pins at the top. The datasheet provides us with the pin layout and its configuration.

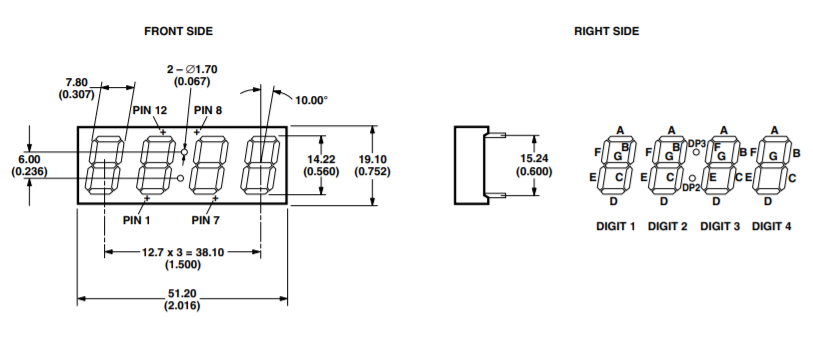


Figure 1: Pin layout

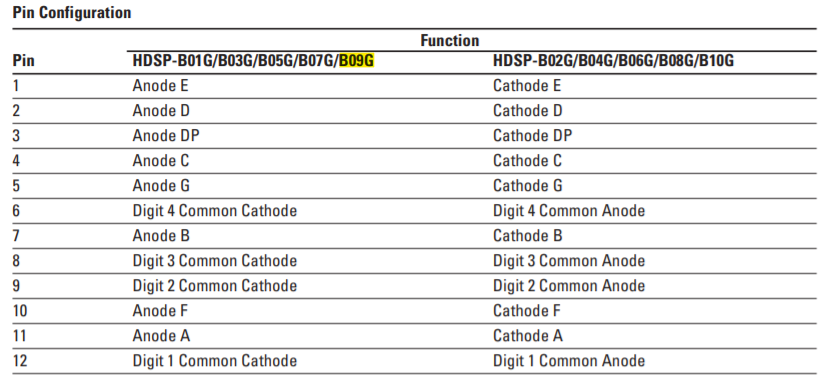


Figure 2: Pin configuration

Each pin of the 7-segment is connected to a separated GPIO-pin of the Raspberry Pi. You can choose which GPIO-pin you use, although this has to be changed in the code that will be shown later on. We chose to connect the GPIO-pins as follows:

|  |  |  |
| --- | --- | --- |
| **7-segment pin** | **GPIO pin** | **Function** |
| 1 | 5 | Anode E |
| 2 | 22 | Anode D |
| 3 | 19 | Anode DP |
| 4 | 27 | Anode C |
| 5 | 13 | Anode G |
| 6 | 25 | Digit 4 Common Cathode |
| 7 | 17 | Anode B |
| 8 | 24 | Digit 3 Common Cathode |
| 9 | 23 | Digit 2 Common Cathode |
| 10 | 6 | Anode F |
| 11 | 4 | Anode A |
| 12 | 18 | Digit 1 Common Cathode |

Figure 3: 7-segment pin with its corresponding GPIO-pin

So, whenever we would like to display the time on the 7-segment display, we would do the following in order: First, we would show the first digit by setting the corresponding GPIO pin (18 in our case) to low (active low). Next, we loop over every segment and set the necessary ones to high (active high). Next, we do the same for every other digit in order. We will go into more detail on the coding later on.

Now that the time is shown on the 7-segment display, we will connect our agenda to the Rapsberry Pi next. In order to do this, we chose to make use of the Google Calendar API. Using this API, we will be able to access (and modify) private events on the user’s calendars. This will be necessary to set the alarm.

# Code

## 7-segment display

In order to control the 7-segment display to show the current time, we will use the GPIO-module provided by the Raspberry Pi. First we initialize the segment-, decimal point- and digit-pins in a tuple as follows:

segments = (4,17,27,22,5,6,13)

dps = (19,)

digits = (18,23,24,25)

The comma after 19 is syntax in Python for indicating that this is in fact a tuple and not just an integer. Afterwards, we will setup and deactivate everything in the 7-segment display by looping over the three tuples:

for segment in segments:

GPIO.setup(segment, GPIO.OUT)

GPIO.output(segment, 0)

for dp in dps:

GPIO.setup(dp, GPIO.OUT)

GPIO.output(dp, 0)

for digit in digits:

GPIO.setup(digit, GPIO.OUT)

GPIO.output(digit, 1)

First, we indicate that every GPIO-pin is an output. Next, we output either a one or a zero to this GPIO-pin. As seen in figure 2 and 3, the segments and decimal point are active high, while the digits are active low. That’s why we output a zero to the segments and decimal point and a one tot the digits. After setting up the GPIO-pins, we will initialize a dictionary with the necessary numbers as follows:

numbers\_dict = {' ':(0,0,0,0,0,0,0),

'0':(1,1,1,1,1,1,0),

'1':(0,1,1,0,0,0,0),

'2':(1,1,0,1,1,0,1),

'3':(1,1,1,1,0,0,1),

'4':(0,1,1,0,0,1,1),

'5':(1,0,1,1,0,1,1),

'6':(1,0,1,1,1,1,1),

'7':(1,1,1,0,0,0,0),

'8':(1,1,1,1,1,1,1),

'9':(1,1,1,1,0,1,1)}

The dictionary can be interpreted as the following. The key is equal to the number we would like to show on the 7-segment display. The value represents a tuple with the segments we should put on active, with the first element being segment A and the last element being segment G. The other segments are in between in logical order. (A, B, C, D, E, F, G) For example: when we would like to display ‘5’ we get the following tuple: (1,0,1,1,0,1,1). When we compare this with the tuple above, we know we should set segments A, C, D, F and G at high, while we set segments B and E at low (active high).

Next, we will finally try to show the current time on the 7-segment display. We will need two for-loop: one for indicating which 7-segment display we would like to control, and the second one for controlling every little LED.

while True:

schedule.run\_pending()

n = time.ctime()[11:13]+time.ctime()[14:16]

s = str(n).rjust(4)

midnightCheck = n+time.ctime()[17:19]

for digit in range(4):

for loop in range(0,7):

GPIO.output(segments[loop],numbers\_dict[s[digit][loop])

if (int(time.ctime()[18:19])%2==0) and (digit==1):

GPIO.output(19, 1)

else:

GPIO.output(19, 0)

GPIO.output(digits[digit], 0)

time.sleep(0.001)

GPIO.output(digits[digit], 1)

First, we start a schedule. This schedule will be used later to schedule an alarm. Next, we

receive the current time via the time-module. Since we only need the hour and minutes, we get

those via array indexing. We convert this into a string with the rjust method with width equal

to 4. Later, we would also like to check whether it is midnight already. That’s why we declare

an extra variable, called ‘midnightCheck’ that stores the hours, minutes and seconds. Now

comes our first for-loop. Because we cannot control the four 7-segment displays concurrently,

we must control them individually. This for-loop is used to control each individual 7-segment

display separately. In the next for-loop we will declare which LED we would like to be brighten

up. Figure 4 shows this in more detail. The first for-loop consists of the first four graphs, while

the inner for-loop consists of the bottom seven graphs.

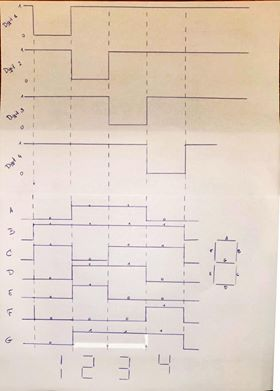


Figure 4: 7-segment display working of code

Furthermore, we let the decimal point blink every second by checking whether the seconds are even or odd.

## Checking agenda and setting alarm

In the next part we will set the correct alarm. This part is only executed when it is midnight. [1]

if(midnightCheck == '000000'):

creds = None

if os.path.exists('token.pickle'):

with open('token.pickle', 'rb') as token:

creds = pickle.load(token)

if not creds or not creds.valid:

if creds and creds.expired and creds.refresh\_token:

creds.refresh(Request())

else:

flow = InstalledAppFlow.from\_client\_secrets\_file('credentials.json', SCOPES)

creds = flow.run\_local\_server(port=0)

with open('token.pickle', 'wb') as token:

pickle.dump(creds, token)

service = build('calendar', 'v3', credentials = creds)

timeMin = datetime.datetime.utcnow().isoformat() + 'Z'

timeMax = (datetime.datetime.utcnow() + datetime.timedelta(hours = 8)).isoformat() + 'Z' #sowieso om 8u opstaan

events\_result = service.events().list(calendarId = 'primary', timeMin = timeMin, timeMax = timeMax, maxResults = 1, orderBy = 'startTime', singleEvents = True).execute()

events = events\_result.get('items', [])

if not events:

alarm\_hour = '08'

alarm\_minute = '00'

for event in events:

start = event['start'].get('dateTime', event['start'].get('date'))

alarm\_hour = str(int(start[11:13]) - 1)

alarm\_minute = start[14:16]

schedule.every().day

alarm\_time = alarm\_hour + ':' + alarm\_minute

schedule.every().day.at(alarm\_time).do(alarm)

First the credentials of the user are checked. Those credentials are stored in the token.pickle-file. When the credentials are not available, the user must login onto his Google-account that stores his agenda. This happens via the Request()-command. Afterwards, the credentials will be valid, and the upcoming event will be requested. The minimum-time is the current time, because it is midnight, while the upcoming events should be ordered by startTime, because we will ask to get the first one in the returned list. As mentioned above, we make use of the schedule-function in Python. So, we update this schedule; the time should be one hour in advance of the upcoming event, while the task should be to wake up the user via an alarm. This alarm is a function, as shown in the code below:

def alarm():

alarm = alarmThread()

alarm.start()

This function calls a class named alarmThread that extends threading.Thread. This is used to create a new thread for the alarm so that the main thread isn’t interrupted when the alarm is playing. In the \_\_init\_\_ function we initialize GPIO 21 as an input this is used to stop the alarm and we initialize pygame.mixer, as shown in the code below:

def \_\_init\_\_(self):

threading.Thread.\_\_init\_\_(self)

self.name = "Alarm thread"

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(21, GPIO.IN)

pygame.mixer.init()

After the initialization we will start playing the alarm using the pygame library. After that we will check if GPIO 21 is still high and the music is still playing and the alarm hasn’t been playing longer than 5 minutes. If one of these conditions is not true any more than the alarm will stop playing, as shown in the code below:

def run(self):

isPlaying = True pygame.mixer.music.load("/home/pi/Documents/Emsys\_project/alarm.wav")

pygame.mixer.music.play()

startTime = datetime.now()

while(isPlaying):

print(GPIO.input(21))

isPlaying = GPIO.input(21) and pygame.mixer.music.get\_busy() and ((startTime - datetime.now()).total\_seconds() < 300)

time.sleep(0.1)

pygame.mixer.music.stop()

Finally, we clean the GPIO of the Raspberry Pi, using the build-in command:

finally:

GPIO.cleanup()

# Future work

* In the future, the Google Assistant API could be implemented so that the user can be informed with some useful information when he is woken up.
* The alarm now goes off one hour in advance. Via the Google Maps API, the Smart Alarm could check the predicted traffic for the next morning and wake the user up accordingly.

# Conclusion

In this project a Raspberry Pi 4 was used to try and build a Smart Alarm. The alarm is capable of checking your personal agenda to set an appropriate alarm. To turn the alarm off the user must push a button, or the alarm will stop playing after 5 minutes.

# Bibliography

|  |  |
| --- | --- |
| [1] | Google, "Python Quickstart," Google, 2019. [Online]. Available: https://developers.google.com/calendar/quickstart/python. [Accessed 9 12 2019]. |
|  |  |