Title: *IoT Smart Alarm Clock*

I. Summary

The smart alarm clock is an open-source project about building and programming an Internet of Things alarm clock. The following project provides all information you need to build the smart alarm clock. The IoT Smart Alarm Clock has text to speech synthesizer, three ways of wake-up sound, set alarm via smartphone or any other computer, running apache2 server, automatic, display brightness adjustment, audio amplifier volume control, 3D-printable case, case built-in tactile switch, alphanumeric display shows text, and 3W speaker definitely wakes you up.

II. Objectives

The purpose of this project is to build an IoT Smart Alarm Clock. The project is designed to help students develop practical IoT device knowledge.

III. Industry-Based Applications

Digital Logic Design is the representation of signals and sequences of a digital circuit through numbers. It is the basis for digital computing and provides a fundamental understanding on how circuits and hardware communicate within a computer. Digital logic is typically implanted into most electronic devices, including calculators, computers, video games, and watches. This project introduces how to connect IoT (internet of things) devices to circuitry. This may relate to a professional engineering job by an engineer may design smart devices. Many engineers may get in this field with the evolution of smart products, such as smart lights, smart homes, smart appliances, smart televisions, smart phones and more.

IV. Project Methodology

A. Components

a. Raspberry Pi Zero (with minor changes you might as well use a different Raspberry Pi)

b. Keyboard, Mouse and Screen (for initial setup)

c. Micro SD Card (8GB or more is recommended)

d. USB-WiFi-Stick (for example: EDIMAX EW-7811UN Wireless USB Adapter)

e. Audio Amplifier (PAM8403)

f. Hole Grid Board (70mm x 50mm x 1.2mm, 24x18 holes, plus one hole in each corner, can be found easily online)

g. Button (Tactile Switch)

h. Photocell (Photoresistor)

i. Speaker (3 Watt 4 Ohm 40mm diameter)

j. 14-Segment Alphanumeric Display

k. Screws: 4x M2 6mm, 4x M2.5 6mm, 4x M2.5 16mm

l. Resistors, Capacitors, Wires

B. Procedure

a. Raspberry Pi Zero Running Raspbian

i. In order to complete this project, you need to set up a Raspberry Pi running Raspbian (Lite), the Debian version of Raspberry Pi. Sure you might use different Raspberry Pis and a different OS, but therefore you might need to modify your system somewhat. For example, if you want to use Raspberry Pi 3 you don't have to buy and use an additional WiFi-stick, might just use the on-board audio output, but might not be able to use the given 3D-printable case. So by changing some parts of this

project you got to keep in mind that this instruction is based on using Raspberry Pi Zero.

Installing Raspbian

ii. First visit the Raspberry Pi downloads page and download the latest Raspbian Lite image. We decided to work with the Lite version, since this project has no need for a graphical user interface. Now follow the instructions page of how to download and write the Raspbian image to your sd-card. Once your sd-card is set up, plug it into your Pi and connect mouse, keyboard and screen to the Pi.

Enabling SSH (Optional)

iii. If students want to control your Pi via remote, navigate to the config menu of the Pi and enable ssh:

```
sudo raspi-config
```

iv. select the Advanced Options menu and press enter. Now select the option SSH and press enable. After sudo rebooting your Pi the changes are applied. By typing:

```
ifconfig
```

v. the Raspberry Pi will tell you its ip-address. From your computer now, being in the same network, you are able to connect to the Pi via ssh like:

```
ssh pi@ip-address
```

vi. enter the password and login to your Pi.

b. Display

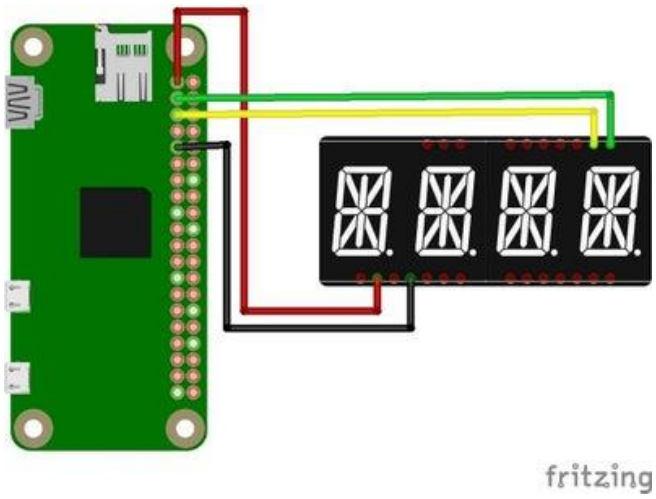

Figure 1- The figure above shows the connections for the display.

i. This step contains the soldering and configuration of the "Alphanumeric Display". Again you might choose a different display. Doing so you have to apply small changes when it comes to coding as well as using the 3d-printable case. The display, we are using for this project, goes by the name "Adafruit 0.54" Quad Alphanumeric FeatherWing Display" and is provided by adafruit. This model comes together with the 14-Segment Alphanumeric LED FeatherWing containing the HT16K33 driver chip, which handles the multiplexing of the commands. Therefore, we are able to control the display using the Raspberry Pi and I²C protocol. And of course adafruit also provide a ready-to-use library, which enables us of simply sending all kinds of strings to the display. But first lets care about the soldering.

Display Assembly

ii. Follow instructions given by adafruit for information about the assembly of the display and the FeatherWing. First you need to solder some pins to the FeatherWing.

iii. Next you need to plug the display into the FeatherWing board and solder all the pins to the board. Again check the adafruit tutorial for details and orientation of the displays assembly.

iv. In order to check the display, you need to connect it to the Raspberry Pi. Therefore, you might want to grab some wires, preferably multiple colors. But before connecting everything together, we recommend to first take the hole grid board and cut out a hole in the lower center. The hole grid board which we are using is 70mm times 50mm and fits perfectly into the provided case. Use the cordless mill to cut a hole into the board, like shown in the pictures above.

v. If you already have printed the case you might take the chance to see if it fits. Put the speaker on the provided cut-out and the hole grid board on top of it. Since those boards are not the highest quality, the position of the four holes at the edges of the board may vary a bit. Don't worry if two of the holes are not fitting exactly to the corresponding screw holes, everything will be squeezed together in the end, such that the use of two screws will be perfectly fine.

vi. Once you are done preparing the hole grid board, soldering the display to the Pi is up next. Grab your colored wires and cut them to a length of about 10cm. Put the display centered into the top row of holes. Check the adafruit tutorials pinout page for details about the wiring. You need to connect four wires: 3.3V; GND and the two I²C pins: SDA and SCL. Consider the schematic diagram concerning the wiring of the display and the Raspberry Pi

vii. Having the display and the raspberry pi successfully soldered and mounted on the hole grid board, your setup should look something like the last picture from above.

viii. Now continue installing the corresponding Adafruit library.

Install Display Library

ix. Once you are done assembling you need to install the Adafruit_LED_Backpack library. First prepare your system by typing:

```
sudo apt-get update
sudo apt-get install build-essential python-dev
```

x. also you'll need to install the python-smbus and python-imaging libraries:

```
sudo apt-get install python-smbus python-imaging
```

xi. In your Raspberry Pis home folder type:

```
git clone https://github.com/adafruit/Adafruit_Python_LED_Backpack.git
```

xii. in order to clone the Adafruit github repository to your Pi. Unzip the file, navigate inside the library's directory and execute the setup.py file:

```
sudo python setup.py install
```

Testing the Display

xiii. Once your installation is finished and your assembly is faultlessly you might go ahead and check if the display is working. First enable the I²C protocol by typing:

```
sudo raspi-config
```

xiv. navigate to the Advanced Options menu and enable the I²C protocol.

```
sudo reboot
```

xv. your Pi to apply the changes. Now switch to the folder

```
cd /Adafruit_Python_LED_Backpack/examples/
```

xvi. and run the following test script:

python alphanum4_test.py

xvii. If you have done everything right, you should be able to see a text message running through the display. Having trouble lighting up all segments you might want to check all your soldering joints again.

c. Clone Project Code

i. In order to test the hardware we are using in the following steps it might be handy to clone the project code of the smart_alarm github repository to your Raspberry Pi first. Make sure you are in the terminal of your Raspberry Pi, whether by using ssh or by simply connecting a screen and keyboard. Establish an internet connection and make sure you are in your home folder by typing cd. Run the following command to install git:

sudo apt-get install git

ii. Next type:

git clone https://github.com/fgebhart/smart_alarm.git

iii. to clone the repository files to your Pi. This might take some minutes. Once everything is downloaded to your Pi go to the folder /home/pi/smart_alarm/tests/ by typing:

cd /home/pi/smart_alarm/tests/

iv. and run the following python script in order to display the time on your alphanumeric display:

python display_time.py

v. Also you might want to run

python scroll_text.py

vi. and play around a little with the given python script. If you should encounter problems using the adafruit Python LED Backpack library try to recap the instructions in section 2. Display and follow the steps in the mentioned adafruit 14-Segment Alpha-numeric LED FeatherWing tutorial to check if you did not forget anything.

d. Step 4 Enable audio

4.1 Low-Pass-Filter Circuit

i. You might have noticed the Raspberry Pi Zeros lack of an audio output. Building an alarm clock without any chance of playing sounds might cause a bunch of oversleeps and angry bosses. But don't worry some skillful people already found a hack to cover the lacking audio output. All you need to do is following the adafruit tutorial of how to Add a Basic Audio Output to Raspberry Pi Zero. Stick to Option 2. Manually Assigning PWM pins for and detailed instruction of how to configure your Raspberry Pi Zero. Here is a short summary of what you need to do. First update your Pi to the latest version:

```
sudo apt-get update
```

ii. Then install the GPIO library:

```
sudo apt-get install raspi-gpio
```

iii. Clone the WiringPi repository to your Pi and install it like described in this tutorial by typing:

```
git clone git://git.drogon.net/wiringPi
```

iv. Once it is successfully downloaded, change directory and build the files:

```
cd ~/wiringPi
```

./build

v. Run:

gpio -v

vi. and

gpio readall

vii. to check if your installation was successful. Like displayed in the readall output you see the function of all pins of your Raspberry Pi Zero:

viii. Change directory to:

cd ~/smart_alarm/tests/

ix. Now compile and install the file gpio_alt.c by typing:

gcc -o gpio_alt gpio_alt.c

sudo chown root:root gpio_alt

sudo chmod u+s gpio_alt

sudo mv gpio_alt /usr/local/bin/

x. Finally we are able to change the alternate functions of the Pis gpio pins. Like mentioned before we are looking to change the functions of pin 13 and 18 in order to enable PWM. Therefore you need to type:

gpio_alt -p 13 -f 0

gpio_alt -p 18 -f 5

xi. Keep in mind, that once you are rebooting your Pi, these functions will be reset to their default function. But don't worry we will cover this concern later by providing a script which is being executed every time on bootup. If you now run the command

gpio readall again, you will notice the new and desired functions of the pins 13 and 18.

xii. Using the alternate function of pins 13 and 18 enables the usage of pulse-width-modulation. Together with a small low-pass-filter circuit we are able to generate a basic audio output. Consider the following schematic diagram for the wiring of the circuit.

xiii. You'll need some resistors and capacitors to build the mentioned low-pass-filter. Of course you might just start soldering the circuit to your hole grid board, but we recommend to first us a breadboard to make sure you connect everything right and test the filter. Note that there are two low-pass-filter, each for one audio output, namely right and left. There is only a need to have one audio output for this project, since there is just one speaker. On the next couple of pictures you will recognize two low-pass-filters, though. This is just because we had some earlier plans of adding two speakers. So don't worry about it and choose what you need for your project. If you are going to build the alarm clock with just one speaker, one filter (=one audio output) is enough. The resistors you need for the low-pass-filter are 150 Ohm and 270 Ohm. The capacitors need to have 10nF and 1µF. Try to find some space on your hole grid board where the resistors and capacitors don't contact the speaker, like shown in the next picture:

xiv. Solder the wires following the fritzing connection diagram above. Once you are done, your circuit should look something like the next picture. The green wires at the bottom are the one coming from Raspberry Pi Zero pins 13 and 18, the black wires at the top of the image are ground and coming from Pi as well. The green wires

at the edge of the board are the audio output coming from the low-pass-filter and will be connected to the speaker and later on to the amplifier in order to increase the volume.

xv. Again note, that there is no need for both low-pass-filter, controlling just one speaker requires only one filter.

Testing Sound

xvi. Once your circuit is soldered and your alternate pin function is configured via device tree overlay, like described in this tutorial you are ready to test your Raspberry Pi Zeros capability of playing sound. Therefore it might be handy to install the omxplayer, by typing: (note: if you did install the Raspbian OS rather than the Lite version, omxplayer is already installed by default)

```
sudo apt-get install omxplayer
```

xvii. now change folder and run omxplayer to play the example audio file:

```
cd ~/smart_alarm/tests/
omxplayer example.mp3
```

Install pygame

xviii. You also need to install the pygame python library in order to play audio files via your python script. Doing so you need to:

```
sudo apt-get install python-pygame
```

xix. Once this is done you should be able to run the following command:

xx. python smart_alarm/tests/play_sound.py The play_sound.py script reads the file in /home/pi/smart_alarm/smart_alarm/music/Festival.mp3 and uses pygame library to play the .mp3 file.

Install Python Text to Speech

xxi. As well as .mp3 files we want to get the alarm clock speaking. Therefore, we need to install the python text to speech package (pyttsx). But first we have to install pip:

```
sudo apt-get install python-pip
```

xxii. Now install pyttsx and the necessary python-espeak library:

```
sudo pip install pyttsx
sudo apt-get install python-espeak
```

xxiii. Again test your system by running:

```
python smart_alarm/tests/say_text.py
```

xxiv. This script prompts you to enter a text and reads it by using pyttsx and espeak.

Install Music Player Client & Daemon

xxv. Because we don't just want to play mp3 files which we have locally on the Pis sd-card, we need to install the Music Player Client & Daemon. Using the music player client (mpc) we are able to listen to tons of different internet radio stations. Since this alarm clock is a Internet of Things alarm clock, we for sure want to enable listening to internet radio stations. So go ahead and install

```
sudo apt-get install mpd mpc
```

xxvi. Now you can use your own client to play the internet radio station you like. Check the most important commands:

xxvii. mpc add radio_station_name : Adds your station to the mpc playlist

xxviii. mpc playlist : Displays the content of the mpc playlist

xxix. mpc clear : Clears the current playlist

xxx. mpc play : plays your playlist, with one radio station at random

xxxi. mpc stop : stops the play of some station

xxxii. In the end we are using those commands via our python scripts together with the python library os like this:

```
os.system('mpc play')
```

xxxiii. But before we are able to run your playlist of radio stations, we first need to add at least one station. Of course there are millions of stations out there and you will find the one you like, but I know you don't want to start searching the web, so I suggest using one of the following two:

xxxiv. http://streaming.radionomy.com/The-Smooth-Lounge?l...

xxxv. http://orange-01.live.sil.at:8000

xxxvi. Depending on your system or where you live you might have to listen to commercials each time you start those stations. So back to our system. Enter the following command to your Raspberry Pi Zeros command line to add the mentioned internet radio station:

```
mpc add  http://orange-01.live.sil.at:8000
```

xxxvii. Now start and stop your playlist in order to check if the music player client is working.

e. Audio Amplifier

i. This section cares about increasing the volume of the system. Again we don't want to cause oversleeping. So adding an amplifier will solve this problem and give your project enough power to even use it like a small jukebox. We choose to use the Stereo Audio Amplifier PAM8403 Chip (see datasheet) since it is cheap and easy to use.

Connecting the Amplifier

ii. Consider the following fritzing schematic for wiring information:

iii. As you can see we are using another Raspberry Pi pin for controlling the amplifier. The yellow wire coming from gpio 12 is connected to the amplifiers switch input pin. Therefore, we need to add another line of code to all future python scripts (don't worry we've already done this!) which enables the amplifier by just sending a logical one for turn on and a zero for turn off vice versa. This appears to be handy since you might have noticed the noise produced by the Raspberry Pis pulse width modulation. Once you have connected everything your setup should look similar to the following one:

Testing the Amplifier

iv. Again run one of the following command in order to test the volume of your system:

omxplayer ~/smart_alarm/test/example.mp3

python ~/smart_alarm/test/play_sound.py

python ~/smart_alarm/test/say_text.py

v. As the sound should be much louder now, you might also want to increase the Raspberry Pis system volume, type:

amixer set PCM -- 100%

vi. which sets the system volume to 100%. If you are having trouble playing sounds, you might want to check if your gpio pins alternate functions are still activated. Once you reboot your system the default function will be activated and you will no longer be able to play sounds via pulse-width-modulation on your Pi Zero. Go back to section 4. Enable Audio. Most likely you just need to reenter the following commands:

gpio_alt -p 13 -f 0

gpio_alt -p 18 -f 5

vii. which we will include in our autostart script later on. Looks like we are done playing sounds. What to do with playing sounds? Stop them!

f. Button

i. How to stop sounds? - Sure by pushing buttons! Once you woke up you need to stop the alarm. Therefore we need to add a button to our project.

Connecting the Button

ii. All of you who have been working with buttons already might know about bouncing and debouncing. If you do not know about it, you might want to read this excellent tutorial by Raspi.TV. If you don't want to read too much, skip the tutorials, grab your button and start wiring. Consider the following fritzing for the wiring:

iii. You might choose some space at the bottom of your hole grid board to solder the connections to. Make sure not to get too close to the speaker. You need two resistors: 1x 1k Ohm and 1x 10k Ohm, one capacitor with 100nF and some wires. Connect everything like shown in the image above. The 1k Ohm resistor is connected between the two red wires, the 10k Ohm resistor is connected between the yellow and black wires. Your soldering on the hole grid board should like the following picture. On the front:

iv. And on the back:

v. If you got your case already printed you might want to check if the button fits into the provided slot like this:

Testing the Button

vi. Like in the previous steps we are going to use a python testing script provided in our project repository. If your button is connected like described above and you already did clone our project repository run the provided python script:

```
python smart_alarm/tests/button_input.py
```

vii. you should see the line button is: 0 printed to your terminal many times. Once you press your button the line changes to button is: 1. If this works, your soldering is correct! If not, check the fritzing diagram from above and recap the mentioned tutorial.

g. Photocell

i. Consider the following fritzing schematic for connecting the photocell to your setup. Note that this graphic is the complete and final wiring of the overall smart alarm project.

ii. Since both photocell and button do not consume much power at all, connecting both to the same power supply pin and ground pin of the Raspberry Pi Zero is possible. You can do this by connecting the photocell wires to the soldering joint at the button on the lower edge of the hole grid board. Your connections should look similar to this one:

iii. Make sure to use wires with a length of about 15cm in order to install the photocell in the top of the case. If the photocell does not fit into the provided hole you might use a sharp knife to slightly enlarge the holes. Take a pen (or some other thin tool) and press the photocell into the flat laying case. The wires can be arranged through the provided slit in the wall between top and body of the case. In the end your installation and wiring should look similar to the following picture:

h. Testing the Photocell

i. Once you are satisfied with the connection of your photocell you might want to check it. Therefore navigate to the /tests folder and run the following python script, by typing:

```
cd ~/smart_alarm/tests/
python reading_brightness.py
```

ii. If you have done everything flawlessly this script should print you your surrounding brightness. Cover the sensor with your fingers and check the data output in your command line.

i. Setup Apache Server

i. One of our goals was to keep the design of the clock as simple as possible. As a result, we had to find a way to configure the clock without the use of more than one button. Therefore we are using an apache webserver on the raspberry pi to set the alarm time, the type of content which is played and many more options.

Installation of Apache Server

ii. To install the apache server on the raspberry pi zero please follow the steps in this tutorial. The plain installation command is:

```
sudo apt-get install apache2 -y
```

iii. Due the fact that we are using python scripts on the server, it is necessary to install the wsgi module for apache:

```
sudo apt-get install libapache2-mod-wsgi
```

iv. Usually the module gets automatically enabled if installed, if not please run the following line:

```
sudo a2enmod wsgi
```

v. Further Information about the wsgi module can be found here.

Configuration of the Apache server

vi. The next step will be the configuration of the apache server. Since apache doesn't have access to the environment variables of the system, we have to declare them separately. Therefore you can add the follwing lines at the end of the file /etc/apache2/envars:

```
# Set smart alarm path variables
export smart_alarm_path=/home/pi/smart_alarm/smart_alarm
```

vii. An example how the file should look like can be found in the git repository under smart_alarm/misc/apache.

viii. After that we can use the variable in the apache config files. The goal is, that our python_server.py script is getting executed, when the server is called in a web browser. The easiest way to achieve this is to replace /etc/apache2/sites-available/000-default.conf with our 000-default.conf under smart_alarm/misc/apache:

```
sudo cp smart_alarm/misc/apache/000-defaul.conf /etc/apache2/sites-available
```

ix. Basically it links the root directory of the server "/" to the python script and it grants the apache server the reading permissions for this folder. The last thing we have to do is giving the apache server the right to change the data.xml file. Usually the apache server has its own user "www-data". There are many different ways to give "www-data" writing permission for the file. The easiest way, but maybe not the securest one, would be to allow everybody to change the file. This can be done in the folder /smart_alarm/smart_alarm/with following lines:

```
sudo chmod o+w data.xml
```

Testing of the Apache server

x. After that you should see the following webpage if you call the ip address or hostname of the raspberry pi in a web browser.

xi. On a PC:

xii. On a mobile phone:

j. Configure Setup

i. In this section we will explain you the final configurations you have to perform in order to get the system working. This includes system configurations, linking the autostart.sh script to the boot process, changing the Pis hostname in order to provide an easy user access and change your Pis user password for more safety.

Autostart Script & Environment Variable

ii. We need the path to the folder containing the python scripts in some places. Therefore we have to define the environment variable “smart_alarm_path”. The standard way is defining the variable in /etc/environment. To do this just add the following line at the end of the file:

smart_alarm_path=”/path/to/smart_alarm/smart_alarm”

iii. The default path would be “/home/pi/smart_alarm/smart_alarm” Besides the apache web server, which should always be running, the smart_alarm script also should be active after booting. To achieve this, the autostart.sh script should be executed automatically after booting. This can be done by adding following two lines at the end of the file /etc/rc.local:

. /etc/environment

. $smart_home_path/autostart.sh

Change Hostname (Optional)

iv. We want to establish a connection to our Raspberry Pi Zero as easy as possible. Therefore we need to change the Pis hostname. Doing so we are able to simply type the Pis hostname in our smartphones (or any other computers) browser. In order to change your Pis hostname we recommend following this tutorial. First you need to edit the following file by typing:

    sudo nano /etc/hosts

v. Here you need to edit the last line which starts with 127.0.1.1. Remove the current hostname raspberrypi and add whatever name you want to give to your system. Now edit the file /etc/hostname by typing:

    sudo nano /etc/hostname

vi. Again replace raspberrypi with the name you want to name your Pi. In order to apply the changes type:

    sudo /etc/init.d/hostname.sh

vii. Since all scripts placed in /init.d/ are being started after each boot up, your hostname will be the name you chose from now on.

Change Password (Optional)

viii. Changing your user password is easy. Even though this is optional, we recommend doing it. Just type:

    sudo raspi-condig

ix. Navigate to the option Change User Password and follow the instructions

k. WiFi Stick

i. In order to fit the Raspberry Pi Zero in the 3D-printable case and save a lot of space, we have to take apart the USB-WiFi-Stick and solder it to the Pis underneath test pads. But first make sure to configure your WiFi connection before soldering it to the board. Therefore you might use a USB hub to provide multiple USB connections.

Configure WiFi

ii. Plug in your WiFi stick and establish a connection to your local network. Once this is done type

ifconfig

iii. on your Pis command line to figure out your ip-address. It turns out ssh is no more activated by default since December 2016. To activate ssh type

sudo raspi-config

iv. navigate to the Advanced Options menu and enable the ssh connection.

sudo reboot

v. your Pi in order to apply the changes. Now try to connect to your Raspberry Pi using your computer and ssh, like this:

ssh pi@ip-address

vi. Replace 'ip-address' in this command by the Pis ip-address or its hostname. Once this connection is established reboot your Pi in order to see if it connects to the configured network automatically. Now shutdown your Pi, unplug all devices and prepare your solder iron.

Take Apart WiFi Stick

vii. Like shown on intro of this instructable, this project uses the "EDIMAX EW-7811UN Wireless USB Adapter". Of course it should be possible to use just any kind of WiFi USB Stick in this project, but keep in mind that we just tested the one described.

viii. Now take apart the WiFi-stick by slightly lifting the plastic case at the flat side of the usb plug. You should be able to remove everything, such that you have the bare wifi board laying in front of you, like shown in the picture

Soldering to Pis Test Pads

ix. After taking apart the WiFi-stick you'll need to cut four wires to a length of about 4cm. Solder the USB connections to the Raspberry Pi Zero test pads like shown in the picture:

x. Keep in mind, that it will not be possible to connect additional USB devices to the Pis micro USB connection after using the USB test pads. Once the soldering is done, plug in the Pis power supply and check if the connection is working. The WiFi-stick should start blinking a few seconds after plugging in the power supply. Wait until booting is done and establish a WiFi ssh connection. All further steps will be based on using a ssh connection like this. In order to check your internet connection and update your Pi to the latest Raspbian version, type:

sudo apt-get update
sudo apt-get upgrade

Configuring Multiple WiFi Connections (Optional)

xi. If you plan to use your device in multiple networks you'll need to configure them first. By just using one WiFi connection you might just skip this step. Adding additional network connections you need to modify the file /etc/wpa_supplicant/wpa_supplicant.conf. Therefore type:

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

xii. and add additional network connections like this.

l. 3D-Printed Case

i. Once you are done building and configuring your setup you are ready to install it to the 3D printed case. The design of the case has been evaluated many times, such that you do not need to print support material at all and in the end all hardware parts fit into the body of the case perfectly.

Download and Print the 3D-Model

ii. See the following link to the 3D model at thingiverse:

http://www.thingiverse.com/thing:2009740

iii. I hope you like the design. We really would appreciate all kinds of improvement ideas, just comment or contact via thingiverse. For 3D-printing the model consider these Cura slicing options.

Assembling the Case

iv. For the installation of the hardware parts in the case and the assembly of the case you need the following screws:

v. 4 x M2 screws length: 6mm (hole grid board)

4 x M2.5 screws length: 6mm (Raspberry Pi Zero)

4 x M2.5 screws length: 16mm (case itself)

vi. After printing the case you might need to remove overlapping material in order to achieve a better fitting. First put the speaker on the provided circular hole. Then try to fit the hole grid board together with the display into the provided slot. You might need to move the display slightly to all sides until it will 'click' into the case. Now put the hole

grid board on top of everything. Make sure to not bend any wires too much when tightening the screws for the hole grid board. Slightly insert the button and the photocell. Afterwards lay down the back part next to the front part and find the right position for the Raspberry Pi Zero. Use the M2.5 screws and mount the Pi Zero to the back of the case. Before closing the case make sure the WiFi board and amplifier are positioned on the side, so you are able to close the case. In the end use the long (16mm) M2.5 screws to tighten the back to the front. Finally, you are done building the smart_alarm project and may start applying the program code to your needs.

m. Getting Started

Refer to the website link for any additional information that is needed.

V. References

[1] Instructables. “IoT Smart Alarm Clock [Open Source Project].” Instructables, Instructables, 21 Sept. 2017, www.instructables.com/id/IoT-Smart-Alarm-Clock-Open-Source-Project/.

Website: https://www.instructables.com/id/IoT-Smart-Alarm-Clock-Open-Source-Project/