The Doorbell project.

Revolutionary doorbell for the convenience and innovation.

By Adam White, Joseph Holland, Teddy Ng and Miroslaw Blicharz

Table of Contents

Project report

Project specification

Project Objectives:

1. Learn design and development of a fully functional and programmable electronic device.
2. Learn project documentation and logging activities.
3. Learn project management and team building soft skills.
4. Learn basic assembly language programming, hardware skills of soldering and moulding box.
5. Learn device design aesthetics.

Expected Outcomes:

1. *Minimum*: Working doorbell activated by push-button. A working doorbell must be able to produce at least one audible tone to a minimum distance of 1 metre from the push-button.
2. *Basic*: The doorbell must be able to produce 4 distinct and discernible short musical pieces. The user can select the music using one or more buttons. The entire hardware for the basic requirement must be housed neatly within the doorbell box provided. The button for activating the doorbell should be appropriately positioned with the speaker.
3. *Enhancements*: The design of the doorbell can be enhanced with added features, so long as the basic requirements are met. These include LCD displays, activation/de-activation remotely, amongst others.

Project plan

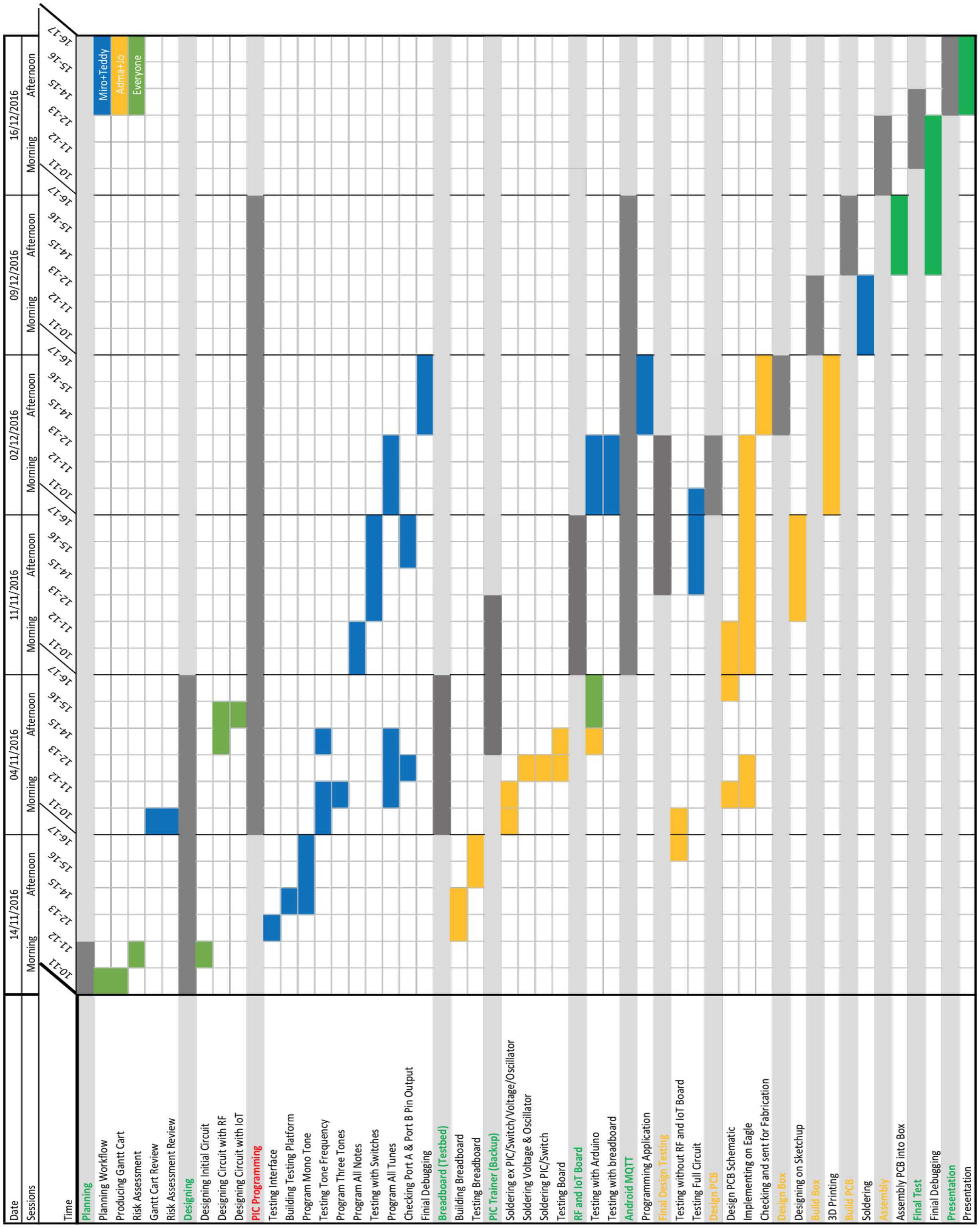


Figure 0 - Gantt Chart

Preliminary design

Schematic diagrams

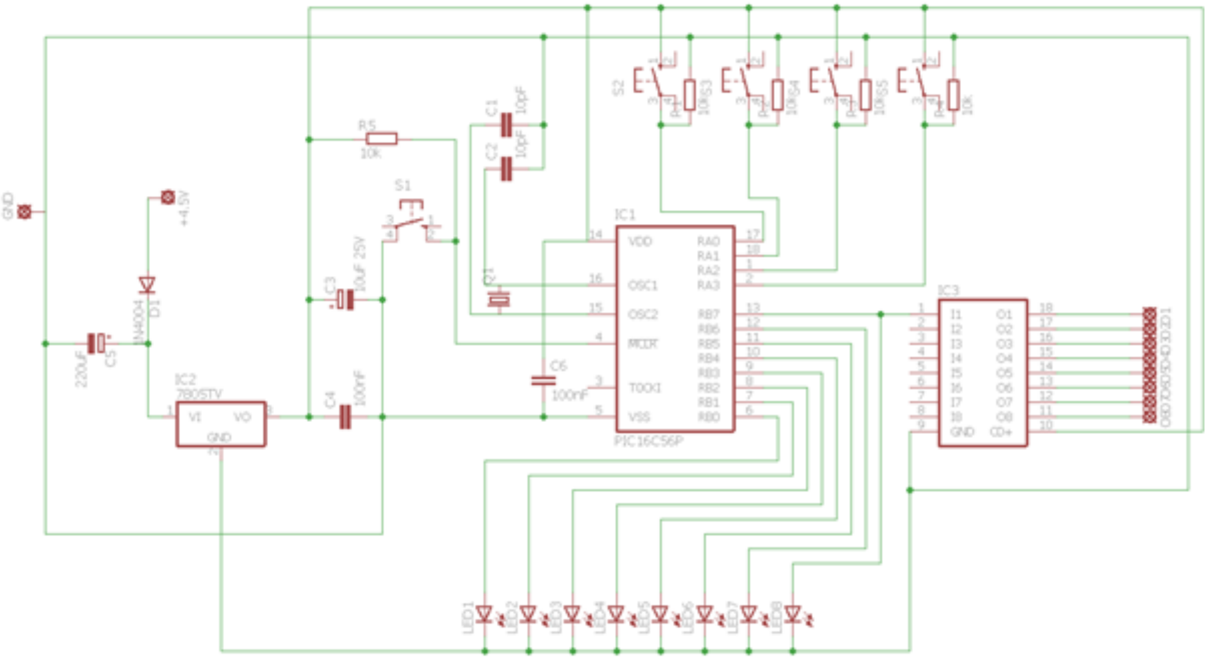


Figure 1 - Initial schematic of the doorbell (Designed in Eagle 7.3.0).

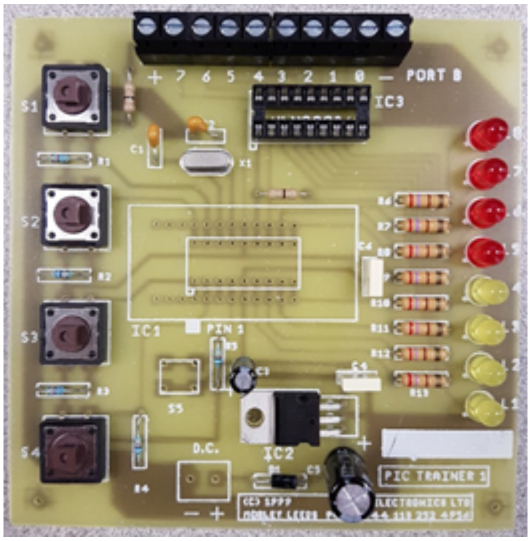


Figure 2 - Initial PCB of the doorbell, created from the schematic in figure 1.

Circuit descriptions

The original schematic was designed with the intention of meeting the basic attributes laid out in the project specification. It allows for a speaker to play a choice of four different tones (coded in assembly) by pressing the switches S2, S3, S4, and S5. The power is supplied by a series of 3 AA batteries with a rating 1.5V each, which then travels through a diode to prevent any backwards flow. The power supply is also in parallel with a decoupling capacitor to remove any potential AC signals which are not wanted in the circuit. The supply then feeds into a voltage regulator to insure no overcharging issues within the circuit. Another decoupling capacitor is placed across VSS and VDD of the PIC to prevent any AC signals between high and low (ground) supplies. S1 is the reset switch of the PIC. Q1 is an oscillator that acts as a clock for the PIC because it does not contain an internal clock. The speaker would connect to one of the ports of the Darlington transistor (IC3) and CD+ which then amplifies the sound and emits it through the speaker. The PIC (IC1) receives input data from the 4 switches and then depending on which input was activated would send data out of pin RB7 to the pin I1 of the Darlington corresponding to one of the 4 sounds programmed. The 8 LEDs on the circuit diagram are connected to each of the RB outputs of the PIC and are used as visual feedback to see if the pins are outputting data.

Although effective, this schematic was very basic and only performed the minimum amount required by the specification and featured no ingenuity or innovation. The initial PCB created was also unnecessarily large with a lot of wasted space.

There was a lot of room for improvement and the ways in which we improved are explained in detail in the documentation package.

Documentation package

Final schematic diagrams

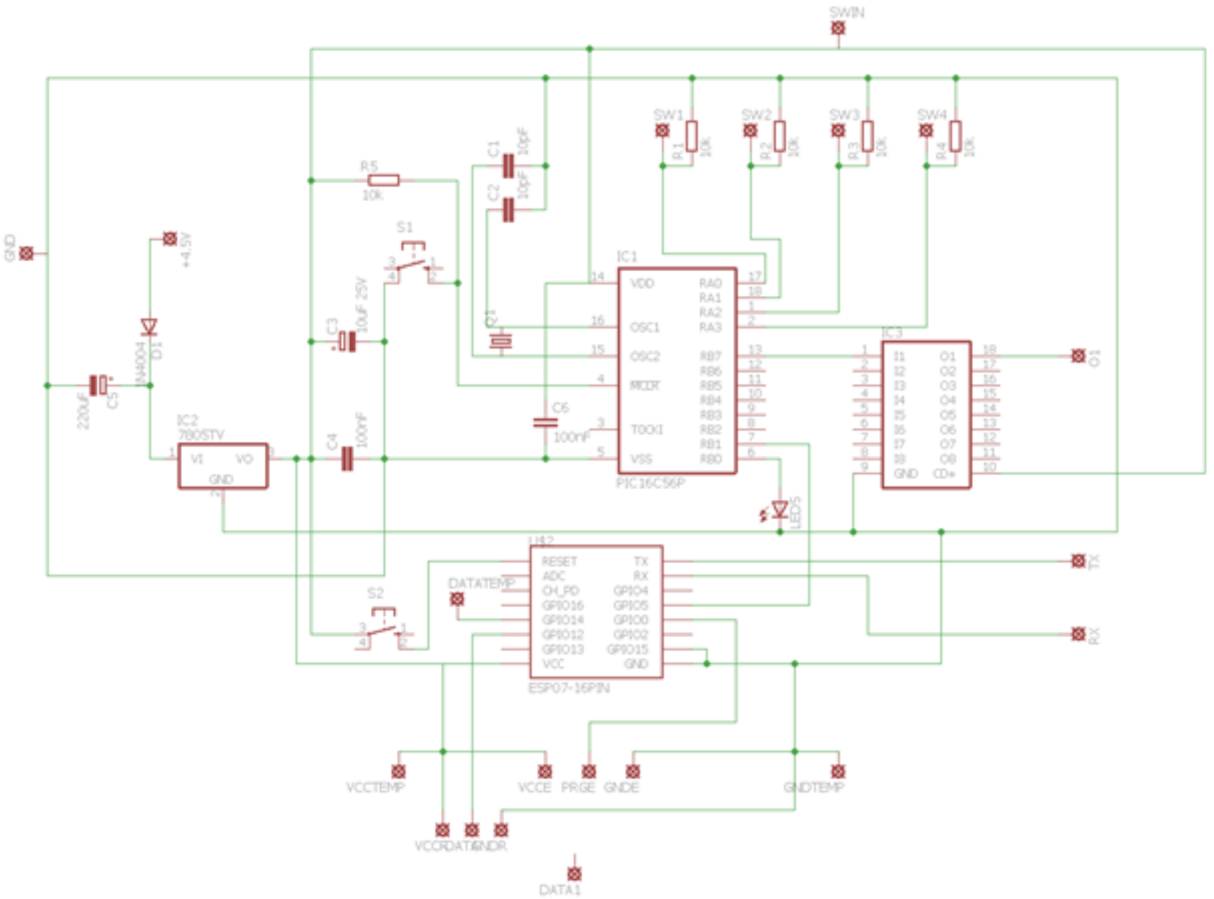


Figure 3 - Final schematic of doorbell receiver board (Designed in Eagle 7.3.0)

To innovate our project we made several changes to the initial schematic. Our most significant change to the board was including a 433Hz receiver so that the doorbell can be activated remotely, and therefore we also needed to create a new board to power the transmitter, shown on the next page. The RF receiver is connected to the pins VCCR, DATA2 and GNDR. It is also connected to DATA1, however this pin serves no purpose and is only included because of the physical design of the receiver.

Another important change we made was the inclusion of an ESP8266-07 microcontroller to give our doorbell Internet of things connectivity. The chip, when receiving a HIGH input from the receiver, emits a signal over Wi-Fi which connects to the cloud and then sends a notification to any connected devices to say that the doorbell has been activated. The chip was coded using the C language and Arduino libraries. The pins VCCE, PRGE and GNDE are connected to allow the ESP chip to be reprogrammed if required, and S2 is the reset switch for the ESP.

To improve on the preliminary design we removed unnecessary components, such as 7 of the 8 LEDS, leaving just one to signify if any signal is being emitted from the pin RB7. (The LED is physically connected to RB0, which is coded to emit HIGH if RB7 is HIGH). We also removed all of the output headers of the Darlington transistor except for one to eliminate any unnecessary clutter.

We also swapped the 4 switches with a single 4 output, 3 pole rotary switch so that only one input could be activated at any one time. This also saved a lot of space on the PCB and allowing a single button activation switch similar to a conventional doorbell.

Also included in the final schematic are two pins VCCTEMP and GNDTEMP, which were intended to be used for a temperature sensor that would also send the temperature of the room to any device connected to the same cloud server as the ESP. However due to time constraints and to avoid over-complication we decided to leave this component out of our final product.

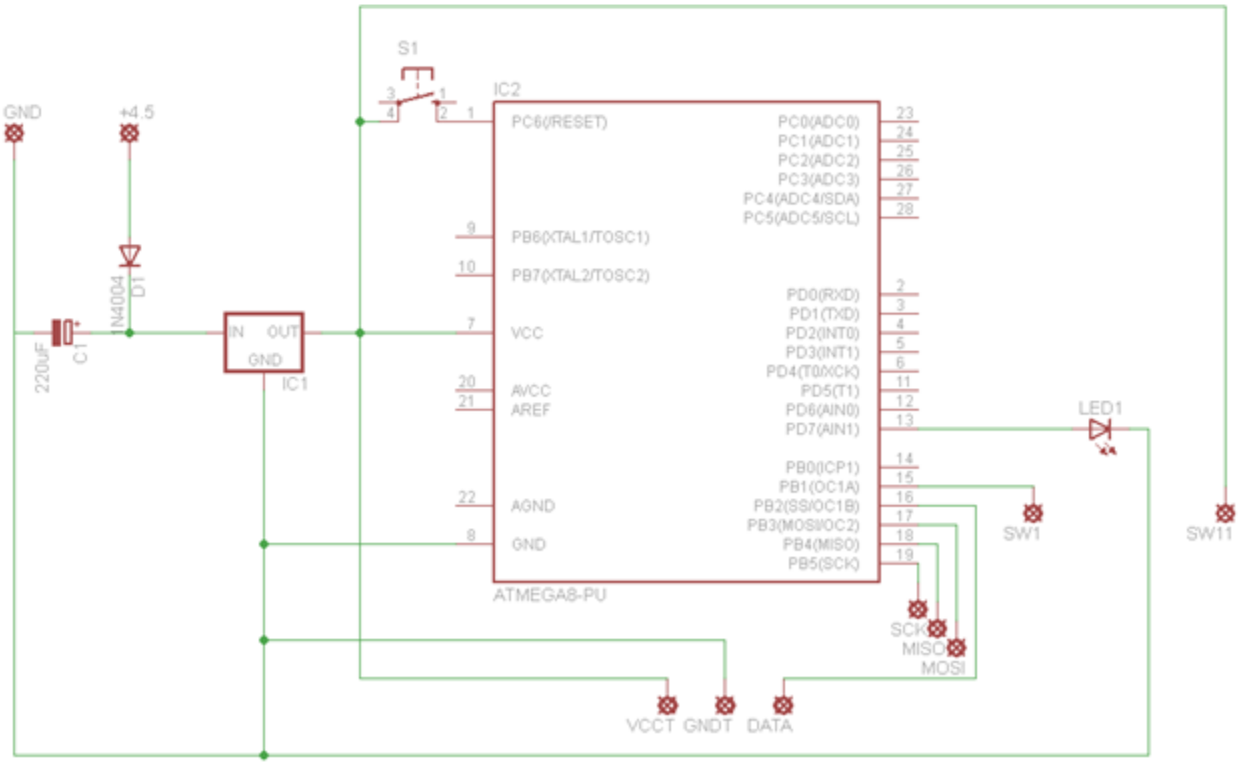


Figure 4 - The final schematic for the doorbell transmitter, aka the button (designed in Eagle 7.3.0)

The board of the transmitter features a 4.5V power supply connected to a diode which blocks any back flow of voltage and a decoupling capacitor that removes any AC signal. The supply then feeds into a 5V voltage regulator to maintain a steady voltage and prevent any unwanted surges. The main component in the transmitter is the ATMEGA8-PU microcontroller which when receiving a HIGH input from the switch SW1-SW11 sends HIGH output through LED1 from PD7 and to DATA, which is the data input pin of the 433Hz transmitter. The pins SCK, MISO and

MOSI are used to reprogram the microcontroller without the need to physically remove the chip from the board.

Printed circuit layout (if applicable)

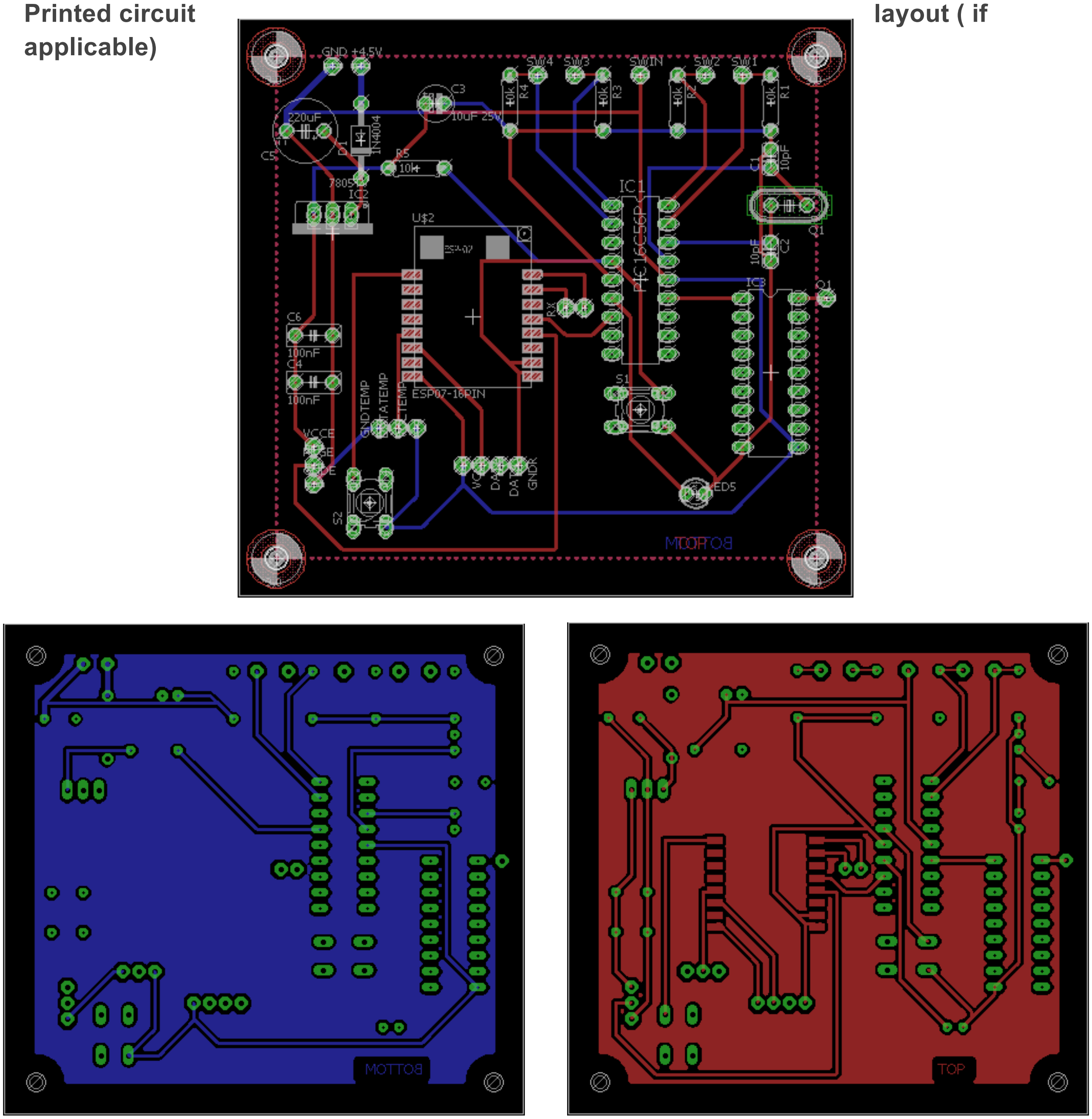


Figure 5 - Final PCB design of receiver, showing All layers, TOP side and BOTTOM side.

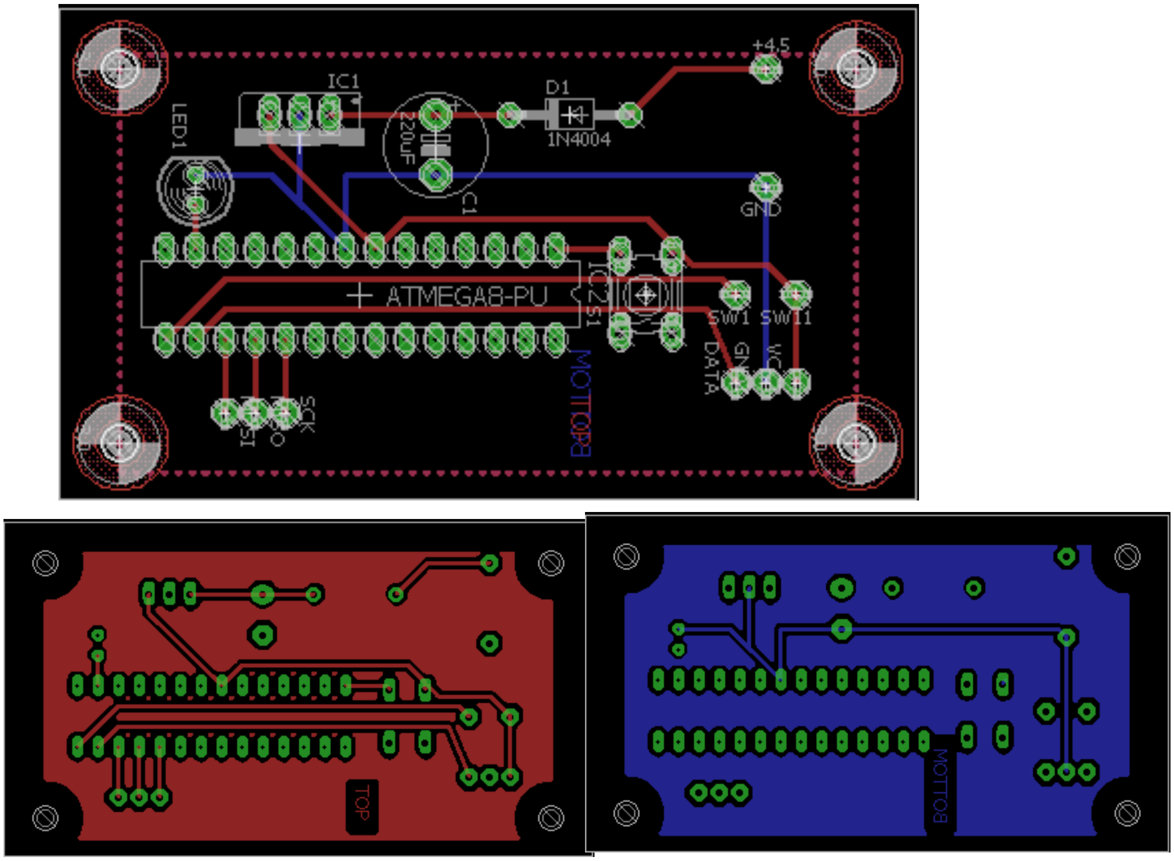


Figure 6 - Final PCB design of transmitter, showing All layers, TOP side and BOTTOM side.

All of the PCBs were designed in Eagle 7.3.0 and were manually routed. We avoided using any vias and therefore crossing tracks were a much more prominent issue in developing the board. After routing all the tracks the drill holes were added to all 4 corners to allow the board to be mounted securely within their respective boxes. Once these were added we used the ratsnest function to create a common plane to allow for easy grounding during the testing of the PCB.

Box Design

Transmitter box:

The transmitter box Is made of laser cut 3mm plastic. It was designed as a net using SketchUp. Then the net was glued together with tabs strengthening the joints. There is a red LED in the box for visual confirmation that the button has been pressed. The removable back held on by pins is hinged creating easy access to the PCB and battery. The bird box design Is to allow for splash resistance and the box has add-on clips to attach it to a wall.

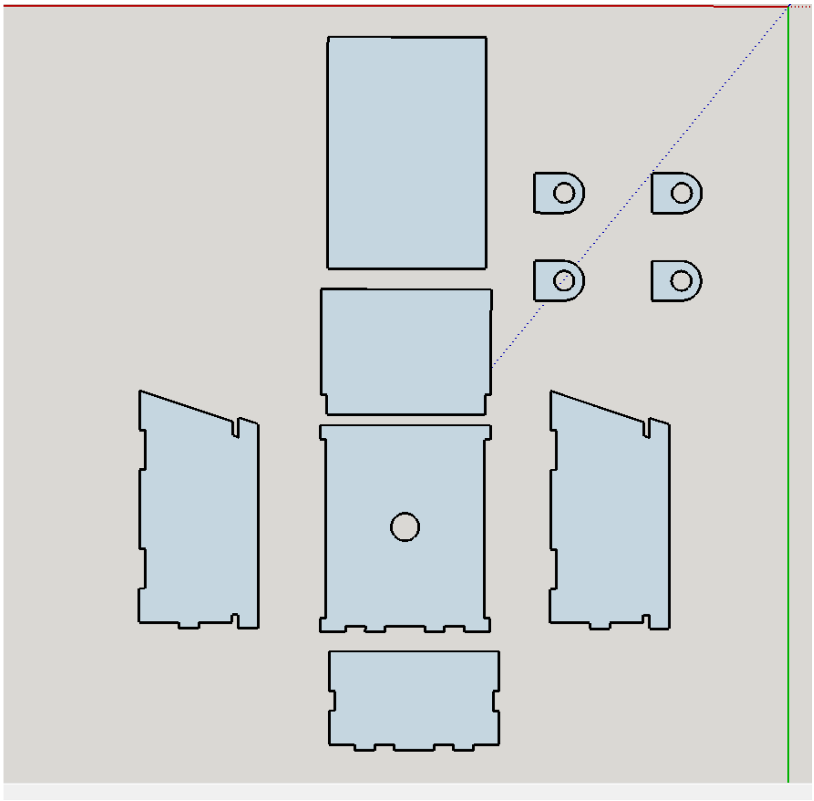


Figure 7 - Net of transmitter box

Receiver Box:

The receiver box is 3D printed with 3mm walls and a battery compartment for a 9V battery and holder. The lid of the box is laser cut with a speaker grill and holes for on/off button and tune selection dial cut into it. Engraved onto the lid are the functions of the buttons. It is held onto the main box by countersunk screws.

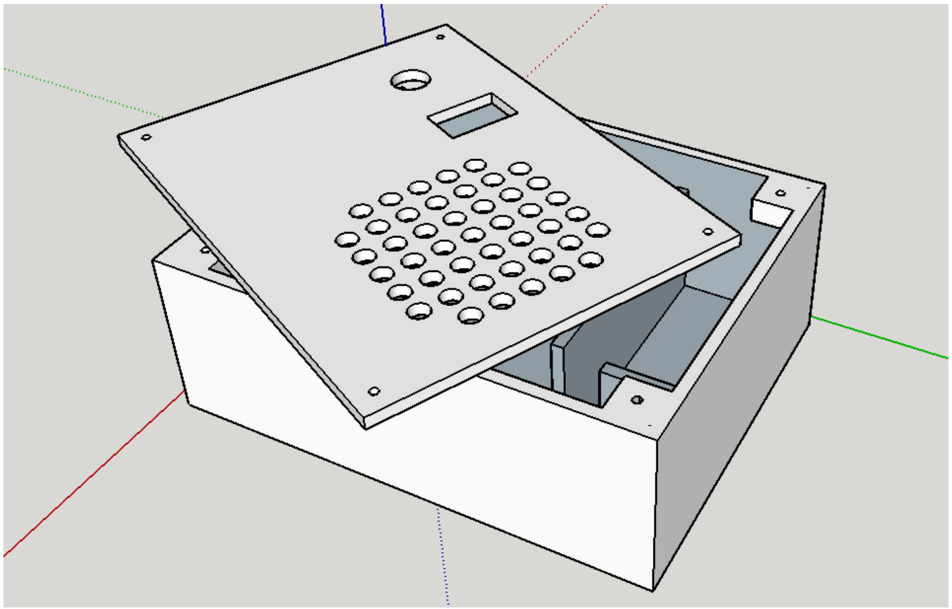


Figure 8 - Receiver box with lid (before countersinking)

Assembly drawings

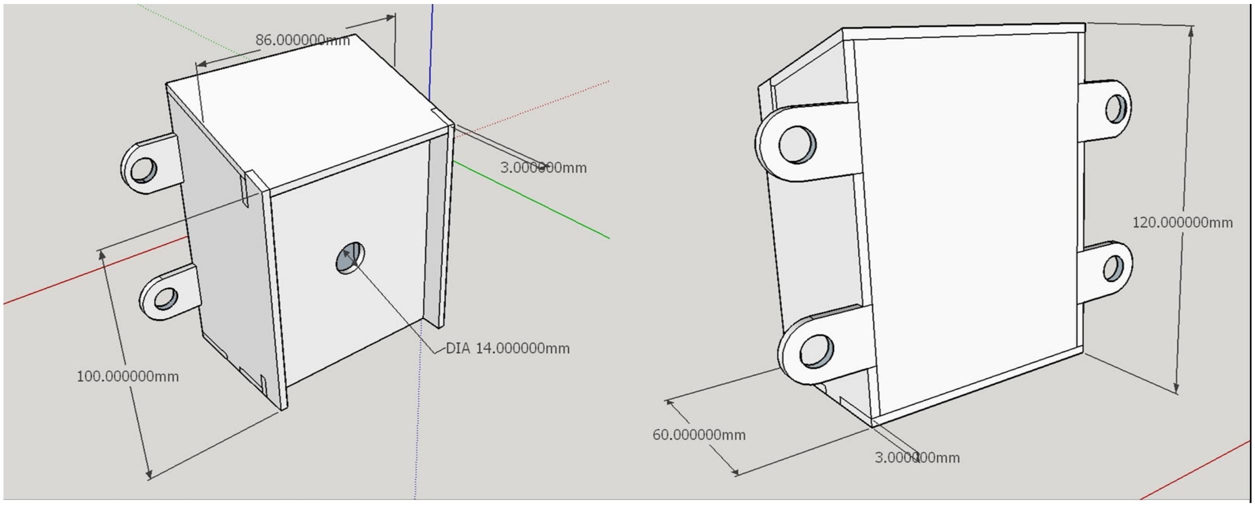


Figure 9 - Final design of transmitter box, aka the button.

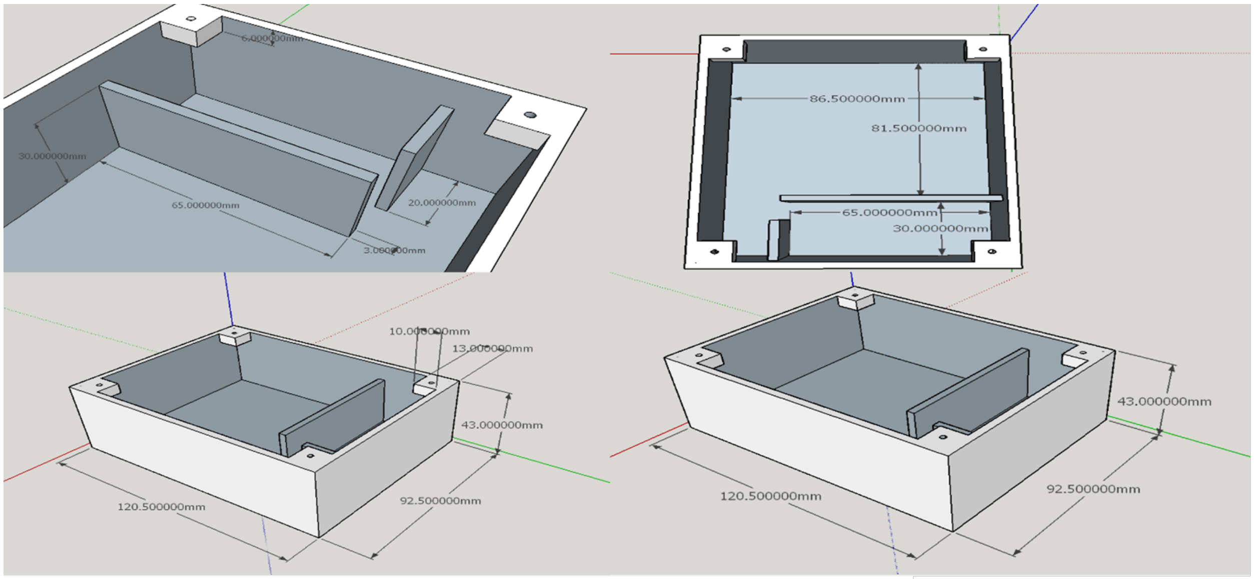


Figure 10 - Final design of receiver box.

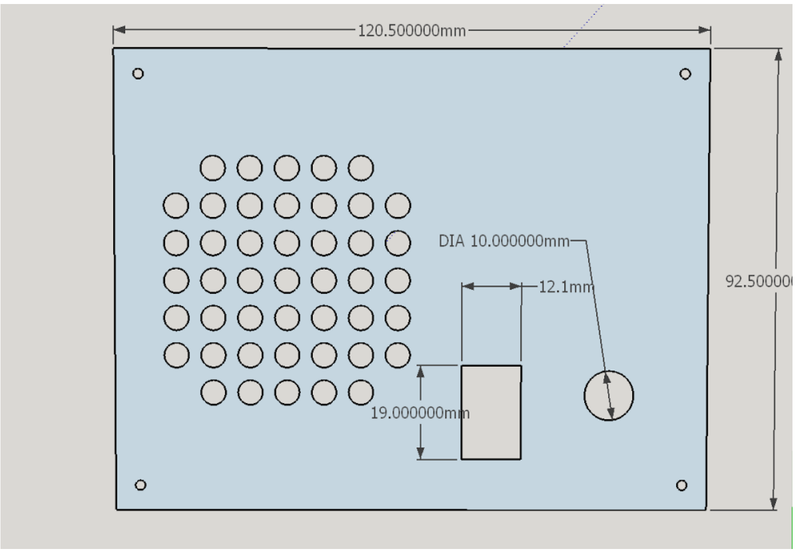


Figure 11 - Final design of lid for the receiver box.

Parts list

The part list has been directly sourced from the EAGLE software we used to design PCBs. It’s a great way to create a parts list as all part names and numbers are listed correctly and will help in identifying components on the PCB.

The doorbell:

Part	Value	Device	Package	Description
C1	10pF	C-EU025-024X044	C025-024X044	CAPACITOR, European symbol
C2	10pF	C-EU025-024X044	C025-024X044	CAPACITOR, European symbol
C3	10uF 25V	CPOL-EUE2.5-5	E2,5-5	POLARIZED CAPACITOR, European symbol
C4	100nF	C-EU050-030X075	C050-030X075	CAPACITOR, European symbol
C5	220uF	CPOL-EUTAP5-80	TAP5-80	POLARIZED CAPACITOR, European symbol
C6	100nF	C-EU050-030X075	C050-030X075	CAPACITOR, European symbol
D1	1N4004	1N4004	DO41-10	DIODE
IC1	PIC16F84A	PIC16F84A	DIL18	MICROCONTROLLER
IC2	7805TV	7805TV	TO220V	Positive VOLTAGE REGULATOR
IC3		ULN2803A	DIL18	DRIVER ARRAY
LED5		LED3MM	LED3MM	LED
Q1		CRYSTALHC49S	HC49/S	CRYSTAL
R1	10k	R-EU_0207/7	0207/7	RESISTOR, European symbol
R2	10k	R-EU_0207/7	0207/7	RESISTOR, European symbol
R3	10k	R-EU_0207/7	0207/7	RESISTOR, European symbol
R4	10k	R-EU_0207/7	0207/7	RESISTOR, European symbol
R5	10k	R-EU_0207/7	0207/7	RESISTOR, European symbol
S1		10-XX	B3F-10XX	OMRON SWITCH
S2		10-XX	B3F-10XX	OMRON SWITCH

U$2	ESP07-16PIN	ESP07-16PIN	ESP07-16PIN	ESP8266 Wifi module 07 - 16 pins

The button:

Part	Value	Device	Package	Description
C1	220uF	CPOL-EUE5-8.5	E5-8,5	POLARIZED CAPACITOR, European symbol
D1	1N4004	1N4004	DO41-10	DIODE
IC1		78XXS	78XXS	VOLTAGE REGULATOR
IC2	ATMEGA8-PU	ATMEGA8-PU	DIL28-3	MICROCONTROLLER
LED1		LED5MM	LED5MM	LED
S1		10-XX	B3F-10XX	OMRON SWITCH

Test at specification

The push button works correctly, when pressed, the tune is played depending on the selected song. The sound is clear and audible for more than just 1 metre, the darlington transistor amplifies the signal very well.

There are four different tunes, made up of musical notes. The program contains all musical notes from C to B of one octave. We have introduced delays so the tunes are more recognisable.

We created two boxes, one for the button and one for the actual doorbell. The boxes are sturdy and well made. Although the button is not connected to the doorbell itself because of our innovation, the button works really well.

Our enhanced features work perfectly, we have tested our RF transmitter and received. We managed to about 25 metres with no obstacles. Through a wall it decreases to about 10-15 metres. The IoT chip handles the RF receiver and sends data to the cloud, informing that the button has been pressed.

Due to the lack of time, we weren’t able to test the battery life on the button and on the doorbell, however the ESP chip is very low power, so it doesn’t draw too much power, and so is the Atmega8. The one that could draw more power is the PIC, however most power will be consumed by the Darlington transistor when it’s operating.

Future improvements

Create an app for the doorbell so you get an actual notification. That could also include a two way communications, e.g. talking to the delivery guy to leave the package in the back garden.

Use interrupts for the Atmega instead of constantly looping waiting for a button - not the best way to do it as it will drain the battery faster, but it was the most reliable at that time and for the prototype.

Include a stop button or a volume control as the doorbell can be a bit annoying and it would be very good to be able to stop it.

Add a temperature sensor to the ESP chip, the pins are already on the PCB, it only needs a DHT22 connected and write up the code in the receiver ESP.

Conclusions

We created a product that met all of the requirements set out in the specification, with a significant amount of innovation (RF connectivity, internet of things connectivity, microcontrollers programmed using C). We also had to create two separate schematics and boards because of the RF components.

We learnt how to code in assembly to make tones for the doorbell, we improved our knowledge of C and used open source libraries to code new components. We designed and developed our own PCBs from beginning to end and then had them printed, afterwards we debugged and resolved several issues with our board using continuity tests.

We developed a Gantt chart in the first week and further improved and expanded it as we went on, and we were always either on or ahead of schedule. We divided our project up into software and hardware teams, which contained two people each, and then further divided tasks into box design, schematic and PCB design, assembly code and C code.

Operating manual

First you must insert a 9V battery into both the transmitter and receiver. On the receiver box you must first unscrew the lid, the battery holder is located below the speaker which must be handled carefully. Move the speaker to the side, insert the battery and then replace the speaker back in its original position.

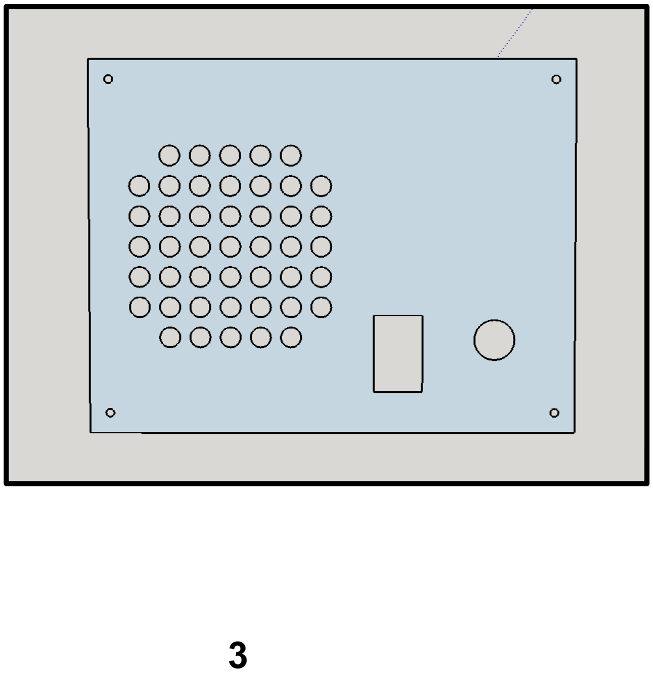
On the transmitter you must remove the top two pins from the back of the box, and then fold out the back on its hinge. The battery holder is located in plain sight and can easily be accessed.

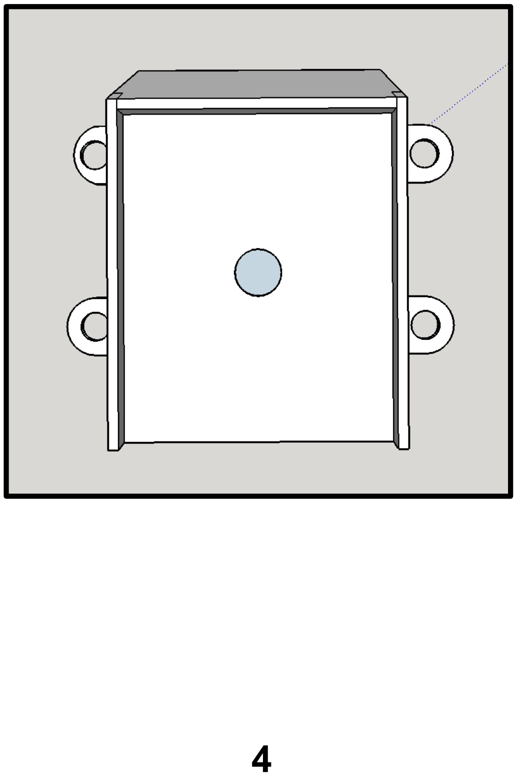
To operate the doorbell the receiver must first be switched on using the power switch located at position **1**. The desired musical tune can be selected using the dial located at position **2**.

You can select from the following 4 tunes:

1. Jingle Bells
2. The Alphabet
3. Old McDonald
4. London Bridge

The sound will be emitted from the speaker at position **3** when the main button on the transmitter box is pressed **(4)**.





1 2

Additional features:

The built in Internet of Things chip allows you to receive notifications on the phone. If you wish to use that feature, bring the box back to the store with your WiFi username and password. After, when you come home make sure to turn it off and on again. Install our app Eric, and follow the instructions on there. You should now be set and should receive notifications on your phone.

Appendix

Software code

Assembly program for the PIC microcontroller

```
        LIST    p=16f84a                ;tell assembler what chip we are using
        include "p16f84a.inc"           ;include the defaults for the chip
        __config _CP_OFF & _WDT_OFF & _XT_OSC ;sets the configuration settings
                                        ;(oscillator type etc.)

LEDPORT Equ     PORTB                   ;set constant LEDPORT to be PORTB
SWPORT  Equ     PORTB                   ;set constant SWPORT = 'PORTB'
SWITCHPORT Equ PORTA
SW1     Equ     0                       ;set constants for the switches
SW2     Equ     1
SW3     Equ     2
SW4     Equ     3
LED1    Equ     0                       ;and for the LED's

        org     0x0000                  ;org sets the origin,
                                        ;this is where the program starts running
        movlw   0x07

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000010'             ;set PortB all outputs
        movwf   TRISB
        movlw   b'00001111'             ;set PortB all outputs
        movwf   TRISA
        bcf     STATUS,         RP0     ;set bank 0
        clrf    PORTB                   ;clear PORTB
```

```
       clrf   PORTA                 ;clear PORTB

Loop                                ;main loop, it waits for a HIGH pin from pin no 4
       btfsc  SWPORT,1              ;which comes from the ESP8266
       call   SwitchMain            ;if it's HIGH, continue to another function which wil
determine the switches.
       goto   Loop

SwitchMain
       call   Del50                 ;give switch time to stop bouncing
       btfsc  SWPORT,7              ;check if the speaker is HIGH, if it is it means it's
already in progress and therefore
                                    ;return back to the initial loop if its working.
       retlw  0x00                  ;return back to the previous loop
       btfss  SWITCHPORT, SW1               ;if switch is low, means it is ON, therefore
proceed to the Switch function
       goto   Switch1

       btfss  SWITCHPORT, SW2               ;if switch is low, means it is ON, therefore
proceed to the Switch function
       goto   Switch2

       btfss  SWITCHPORT, SW3               ;if switch is low, means it is ON, therefore
proceed to the Switch function
       goto   Switch3

       btfss  SWITCHPORT, SW4               ;if switch is low, means it is ON, therefore
proceed to the Switch function
       goto   Switch4

Tune1                 ;function for playing the specific tune, this one is for Switch1
                      ;we're calling the note function note by note.
       call   ELoop
       call   ELoop
       call   ELoopL

       call   Del10
       call   ELoop
       call   ELoop
       call   ELoopL
       call   Del10

       call   ELoop
       call   GLoop
       call   CLoop
       call   DLoopS
       call   ELoopL
       call   Del20

       call   FLoop
       call   FLoop
       call   FLoopL
       call   FLoopS
       call   FLoop
       call   ELoop
       call   ELoop

       call   ELoopS
       call   ELoopS
       call   ELoop
       call   DLoop
       call   DLoop

       call   ELoop
       call   DLoopL
       call   GLoopL
       call   Del20

       call   ELoop
       call   ELoop
       call   ELoopL
       call   Del10

       call   ELoop
       call   ELoop
       call   ELoopL
       call   Del10

       call   ELoop
       call   GLoop
       call   CLoop
       call   DLoopS
       call   ELoopL
       call   Del20

       call   FLoop
       call   FLoop
       call   FLoopL

       call   FLoopS
       call   FLoop
       call   ELoop
       call   ELoop

       call   ELoopS
       call   ELoopS
       call   GLoop
       call   GLoop

       call   FLoop
       call   DLoop
       call   CLoopL
       return         ;after its finished, return back to the Switch function

Tune2                 ;function for playing the specific tune, this one is for Switch2
```

```
               ;we're calling the note function note by note.
call   CLoop
call   CLoop
call   GLoop
call   GLoop
call   ALoop
call   ALoop
call   GLoopL

call   Del10
call   FLoop
call   FLoop
call   ELoop
call   ELoop

call   DLoop
call   DLoop
call   CLoopL

call   Del20

call   GLoop
call   GLoop
call   FLoop
call   FLoop

call   ELoop
call   ELoop
call   DLoopL

call   Del10

call   GLoop
call   GLoop
call   FLoop
call   FLoop

call   ELoop
call   ELoop
call   DLoopL

call   Del20

call   CLoop
call   CLoop
call   GLoop
call   GLoop

call   ALoop
call   ALoop
call   GLoopL

call   FLoop
```

```
        call    FLoop
        call    ELoop
        call    ELoop

        call    DLoop
        call    DLoop
        call    CLoopL

        return

Tune3                 ;function for playing the specific tune, this one is for Switch3
                      ;we're calling the note function note by note.

        call    GLoop
        call    GLoop
        call    GLoop
        call    DLoop

        call    ELoop
        call    ELoop
        call    DLoopL

        call    Del20

        call    BLoop
        call    BLoop
        call    ALoop
        call    ALoop
        call    GLoopL

        call    DLoop
        call    GLoop
        call    GLoop
        call    GLoop
        call    DLoop

        call    ELoop
        call    ELoop
        call    DLoopL

        Call    Del20

        call    BLoop
        call    BLoop
        call    ALoop
        call    ALoop
        call    GLoopL

        return

Tune4                 ;function for playing the specific tune, this one is for Switch4
                      ;we're calling the note function note by note.
        call    GLoopL
        call    ALoopS
        call    GLoop
        call    FLoop

        call    ELoop
        call    FLoop
        call    GLoopL

        call    Del10

        call    DLoop
        call    ELoop
        call    FLoopL

        call    Del10

        call    ELoop
        call    FLoop
        call    GLoopL

        call    Del10

        call    GLoopL
        call    ALoopS
        call    GLoop
        call    FLoop

        call    ELoop
        call    FLoop
        call    GLoopL

        call    Del10

        call    DLoopL
        call    GLoopL

        call    ELoop
        call    CLoop

        return

Switch1 call    Del50                 ;give switch time to stop bouncing
        btfsc   SWITCHPORT, SW1               ;check if switch is high
        retlw   0x00                  ;if it is, return
        btfss   SWPORT,7              ;check if speaker is OFF
        goto    Tune1                 ;if it is, play the tune

Switch2 call    Del50                 ;give switch time to stop bouncing
        btfsc   SWITCHPORT, SW2               ;check if switch is high
        retlw   0x00                  ;if it is, return
        btfss   SWPORT,7              ;check if speaker is OFF
        goto    Tune2                 ;if it is, play the tune
```

```
Switch3 call    Del50                 ;give switch time to stop bouncing
        btfsc   SWITCHPORT, SW3               ;check if switch is high
        retlw   0x00                  ;if it is, return
        btfss   SWPORT,7              ;check if speaker is OFF
        goto    Tune3                 ;if it is, play the tune

Switch4 call    Del50                 ;give switch time to stop bouncing
        btfsc   SWITCHPORT, SW4               ;check if switch is high
        retlw   0x00                  ;if it is, return
        btfss   SWPORT,7              ;check if speaker is OFF
        goto    Tune4                 ;if it is, play the tune

LED1ON  bsf     LEDPORT, LED1  ;turn LED1 on
        call    Del50
        btfsc   SWPORT,SW1     ;wait until the s
        retlw   0x00
        goto    LED1ON


        cblock  0x20                  ;start of general purpose registers
                count1                ;used in delay routine
                counta                ;used in delay routine
                countb                ;used in delay routine
                D1                    ;used in Tune routine
                D2                    ;used in Tune routine
                J                     ;used in Tune lenght routine
                K                     ;used in Tune lenght routine
        endc

CLoop
        movlw d'80'
        movwf J
a1loop:
        movwf K
        movlw   0xff
        movwf   PORTB
        nop                           ;the nop's make up the time taken by the goto
        nop                           ;giving a square wave output
        call    CNote                 ;this waits for a while!
        movlw   0x00
        movwf   PORTB                 ;set all bits off
        call    CNote
a2loop:
        decfsz K,f
        goto a2loop
        decfsz J,f
        goto a1loop

        return

DLoop
        movlw d'80'
        movwf J
```

```
b1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   DNote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   DNote
b2loop:
        decfsz K,f
        goto b2loop
        decfsz J,f
        goto b1loop

        return

ELoop
        movlw d'80'
        movwf J
c1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   ENote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   ENote
c2loop:
        decfsz K,f
        goto c2loop
        decfsz J,f
        goto c1loop
        return

FLoop
        movlw d'80'
        movwf J
d1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   FNote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   FNote
d2loop:
        decfsz K,f
        goto d2loop
        decfsz J,f
        goto d1loop

        return

GLoop
        movlw d'80'
        movwf J
e1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   GNote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   GNote
e2loop:
        decfsz K,f
        goto e2loop
        decfsz J,f
        goto e1loop

        return

ALoop
        movlw d'100'
        movwf J
f1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   ANote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   ANote
f2loop:
        decfsz K,f
        goto f2loop
        decfsz J,f
        goto f1loop
        return

BLoop
        movlw d'100'
        movwf J
g1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   BNote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   BNote
g2loop:
        decfsz K,f
        goto g2loop
        decfsz J,f
        goto g1loop
        return

CLoopL
        movlw d'140'
        movwf J
h1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   CNote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   CNote
h2loop:
        decfsz K,f
        goto h2loop
        decfsz J,f
        goto h1loop

        return

DLoopL
        movlw d'140'
        movwf J
i1loop:
        movwf K
        movlw  0xff
        movwf  PORTB
        nop                          ;the nop's make up the time taken by the goto
        nop                          ;giving a square wave output
        call   DNote                 ;this waits for a while!
        movlw  0x00
        movwf  PORTB                 ;set all bits off
        call   DNote
i2loop:
        decfsz K,f
        goto i2loop
        decfsz J,f
        goto i1loop
        return

ELoopL
        movlw d'140'
        movwf J
j1loop:
        movwf K
        movlw   0xff
        movwf   PORTB
        nop                         ;the nop's make up the time taken by the goto
        nop                         ;giving a square wave output
        call    ENote               ;this waits for a while!
        movlw   0x00
        movwf   PORTB               ;set all bits off
        call    ENote
j2loop:
        decfsz K,f
        goto j2loop
        decfsz J,f
        goto j1loop
        return

FLoopL
        movlw d'140'
        movwf J
k1loop:
        movwf K
        movlw   0xff
        movwf   PORTB
        nop                         ;the nop's make up the time taken by the goto
        nop                         ;giving a square wave output
        call    FNote               ;this waits for a while!
        movlw   0x00
        movwf   PORTB               ;set all bits off
        call    FNote
k2loop:
        decfsz K,f
        goto k2loop
        decfsz J,f
        goto k1loop

        return

GLoopL
        movlw d'140'
        movwf J
l1loop:
        movwf K
        movlw   0xff
        movwf   PORTB
        nop                         ;the nop's make up the time taken by the goto
        nop                         ;giving a square wave output
```

```
       call   GNote                 ;this waits for a while!
       movlw  0x00
       movwf  PORTB                 ;set all bits off
       call   GNote
l2loop:
       decfsz K,f
       goto l2loop
       decfsz J,f
       goto l1loop

       return

DLoopS
       movlw d'40'
       movwf J
m1loop:
       movwf K
       movlw  0xff
       movwf  PORTB
       nop                          ;the nop's make up the time taken by the goto
       nop                          ;giving a square wave output
       call   DNote                 ;this waits for a while!
       movlw  0x00
       movwf  PORTB                 ;set all bits off
       call   DNote
m2loop:
       decfsz K,f
       goto m2loop
       decfsz J,f
       goto m1loop

       return

ELoopS
       movlw d'40'
       movwf J
n1loop:
       movwf K
       movlw  0xff
       movwf  PORTB
       nop                          ;the nop's make up the time taken by the goto
       nop                          ;giving a square wave output
       call   ENote                 ;this waits for a while!
       movlw  0x00
       movwf  PORTB                 ;set all bits off
       call   ENote
n2loop:
       decfsz K,f
       goto n2loop
       decfsz J,f
       goto n1loop
       return
```

```
FLoopS
        movlw d'40'
        movwf J
o1loop:
        movwf K
        movlw   0xff
        movwf   PORTB
        nop                         ;the nop's make up the time taken by the goto
        nop                         ;giving a square wave output
        call    FNote               ;this waits for a while!
        movlw   0x00
        movwf   PORTB               ;set all bits off
        call    FNote
o2loop:
        decfsz K,f
        goto o2loop
        decfsz J,f
        goto o1loop

        return

ALoopS
        movlw d'40'
        movwf J
p1loop:
        movwf K
        movlw   0xff
        movwf   PORTB
        nop                         ;the nop's make up the time taken by the goto
        nop                         ;giving a square wave output
        call    ANote               ;this waits for a while!
        movlw   0x00
        movwf   PORTB               ;set all bits off
        call    ANote
p2loop:
        decfsz K,f
        goto p2loop
        decfsz J,f
        goto p1loop
        return

CNote
                     ;3803 cycles
        movlw   0xF8
        movwf   D1
        movlw   0x03
        movwf   D2
CNote_0
        decfsz  D1, f
        goto    $+2
        decfsz  D2, f
        goto    CNote_0
                     ;3 cycles
        goto   $+1
        nop
                       ;4 cycles (including call)
        return


DNote
                       ;3393 cycles
        movlw  0xA6
        movwf  D1
        movlw  0x03
        movwf  D2
DNote_0
        decfsz D1, f
        goto   $+2
        decfsz D2, f
        goto   DNote_0
                       ;4 cycles
        goto   $+1
        goto   $+1
                       ;4 cycles (including call)
        return

ENote
                       ;3023 cycles
        movlw  0x5C
        movwf  D1
        movlw  0x03
        movwf  D2
ENote_0
        decfsz D1, f
        goto   $+2
        decfsz D2, f
        goto   ENote_0
                       ;3 cycles
        goto   $+1
        nop
                       ;4 cycles (including call)
        return

FNote
                       ;2858 cycles
        movlw  0x3B
        movwf  D1
        movlw  0x03
        movwf  D2
FNote_0
        decfsz D1, f
        goto   $+2
        decfsz D2, f
        goto   FNote_0
                       ;3 cycles
        goto   $+1
        nop
                        ;4 cycles (including call)
        return

GNote
                        ;2543 cycles
        movlw   0xFC
        movwf   D1
        movlw   0x02
        movwf   D2
GNote_0
        decfsz  D1, f
        goto    $+2
        decfsz  D2, f
        goto    GNote_0
                        ;3 cycles
        goto    $+1
        nop
                        ;4 cycles (including call)
        return

ANote
                        ;2263 cycles
        movlw   0xC4
        movwf   D1
        movlw   0x02
        movwf   D2
ANote_0
        decfsz  D1, f
        goto    $+2
        decfsz  D2, f
        goto    ANote_0
                        ;3 cycles
        goto    $+1
        nop
                        ;4 cycles (including call)
        return

BNote
                        ;2013 cycles
        movlw   0x92
        movwf   D1
        movlw   0x02
        movwf   D2
BNote_0
        decfsz  D1, f
        goto    $+2
        decfsz  D2, f
        goto    BNote_0
                        ;3 cycles
        goto    $+1
        nop
                        ;4 cycles (including call)
```

```
       return

       ;//////////// GENERAL DELAYS ////////////////
Del0   retlw  0x00                   ;delay 0mS - return immediately
Del1   movlw  d'1'                   ;delay 1mS
       goto   Delay
Del5   movlw  d'5'                   ;delay 5mS
       goto   Delay
Del10  movlw  d'10'                  ;delay 10mS
       goto   Delay
Del20  movlw  d'20'                  ;delay 20mS
       goto   Delay
Del50  movlw  d'50'                  ;delay 50mS
       goto   Delay
Del100 movlw  d'100'                 ;delay 100mS
       goto   Delay
Del250 movlw  d'250'                 ;delay 250 ms

Delay  movwf  count1
d1     movlw  0xC7                   ;delay 1mS
       movwf  counta
       movlw  0x01
       movwf  countb
Delay_0
       decfsz counta, f
       goto   $+2
       decfsz countb, f
       goto   Delay_0

       decfsz count1 ,f
       goto   d1
       retlw  0x00
       end
```

C program for the doorbell

```
/* Miroslaw Blicharz Semester 1 2016/17
   Incorporating MQTT protocol with a 433Mhz RF receiver.
   Used for the doorbell project
 */

// including some headers that will be used for the receiver and esp chip
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <MQTT.h>
#include <RCSwitch.h>

RCSwitch mySwitch = RCSwitch(); // initialising new RC switch

const char *ssid =  "AndroidAP";    // cannot be longer than 32 characters!
const char *pass =  "xhct2880";   // insert your internet SSID and password
const int output = 4; // output that will drive the PIC high or low
```

```
WiFiClient wclient; //initialising new WifiClient
PubSubClient client(wclient, "m21.cloudmqtt.com", 18694);

void setup() { //in the setup we initialise the RC switch to a specific pin (12) and setting
output as output pin.
  Serial.begin(115200);
  delay(10);
  mySwitch.enableReceive(12);
  pinMode(output, OUTPUT);
  digitalWrite(output, LOW);
}

void loop() {

  if (WiFi.status() != WL_CONNECTED) { // first we want to connect to the wifi
    Serial.printf("Connecting to ");
    Serial.printf(ssid);
    Serial.println("...");
    WiFi.begin(ssid, pass);

    if (WiFi.waitForConnectResult() != WL_CONNECTED) //go back to the loop if not connected.
      return;
    Serial.println("WiFi connected");

  }
  if (WiFi.status() == WL_CONNECTED) { //if connected, now its time to connect to the MQTT
    if (!client.connected()) {
      if (client.connect(MQTT::Connect("arduinoClient")
                         .set_auth("scknjdou", "d44gkmL5YXx5"))) {
        Serial.println("Connected to MQTT server");
      }
    }

    if (mySwitch.available()) { // if the receiver is ready to receive the signal

      int value = mySwitch.getReceivedValue();

      if (value == 0) {
        Serial.print("Unknown encoding");
      } else {
        Serial.print("Received ");
        Serial.println( mySwitch.getReceivedValue() );
        Serial.print("Delay ");
        Serial.println( mySwitch.getReceivedDelay() ); //purely for debugging reason, there is
no need to print it to serial.

        if (mySwitch.getReceivedValue() == 1) { //if I receive a one from the transmitter, i
want to send a HIGH pin to the PIC so it can play a tune.
          digitalWrite(output, HIGH);
          Serial.println("Message sent");
          client.publish("test/Topic2", "Start");
        }
        else { // however if its not 1 then I want the PIC to be constantly OFF, therefore
writing it low.
          digitalWrite(output, LOW);
          client.publish("test/Topic2", "Stop");
        }
      }
      delay(300);
      digitalWrite(output, LOW); //making sure the PIC stays low after receiving the "1".
      mySwitch.resetAvailable(); //RC switch is available to listen to the signal again.
    }

    if (client.connected()) //if I'm connected, continue waiting for either publish or
subscribe
      client.loop();
  }
}
```

C program for the button

```
/* Miroslaw Blicharz Semester 1 2016/17
   Incorporating MQTT protocol with a 433Mhz RF receiver.
   Used for the doorbell project
*/

#include <RCSwitch.h> //this connects to the library of the RC switch used for the transmitter

RCSwitch mySwitch = RCSwitch(); //initialising new RC switch

int buttonPin = 9; //specifying button and LED pin
int ledPin = 7;

int buttonState = 0;         // current state of the button
int lastButtonState = 0;     // previous state of the button
// the follow variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long time_down = 0;         // the last time the output pin was toggled
long debounce = 300;   // the debounce time, increase if the output flickers

void setup() {
  Serial.begin(9600);
  // Transmitter is connected to Atmega Pin #10
  mySwitch.enableTransmit(10);


  // Optional set protocol (default is 1, will work for most outlets)
  mySwitch.setProtocol(1);
  // Optional set pulse length.
  //mySwitch.setPulseLength(140);

  // Optional set number of transmission repetitions.
  //mySwitch.setRepeatTransmit(10);

  // initialize the button pin as a input:
```

```
  pinMode(buttonPin, INPUT_PULLUP);
  // initialize the LED as an output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the pushbutton input pin:
  buttonState = digitalRead(buttonPin);

  if (buttonState == LOW && millis() - time_down > debounce) {
    // if the pushbutton has been pressed, write HIGH the LED pin
    digitalWrite(ledPin, HIGH);
    // send a value of 1 using 8 bits (doesnt really matter what number, but be careful about
the number of bits
    mySwitch.send(1, 8);
    delay(3000);   //wait 3 seconds before turning the LED off
    digitalWrite(ledPin, LOW);
    time_down = millis();
  }
}
```