arm Education

Lab 7

GPIO PROJECT:

SPEAKER OUTPUT

Contents

1 Introduction

1.1 Lab overview

For this project, you will create a device which makes sounds through a speaker based on how the user presses several switches.

2 Requirements

In this lab, we will be using the following hardware and software:

- **Keil µVision5 MDK IDE**
  - Please see the included Getting Started with Keil guide on how to download and install Keil.
- **STM32 Nucleo-L552ZE-Q**
  - For more information, click here.
- **Logic Analyzer or Oscilloscope**
  - Required to monitor various signals
- **3x Switches/Buttons**
  - **Includes full circuit for these (the buttons themselves, 100k Ω Resistors, etc.)**
- **Speaker module**
- **330 Ω Resistor**
- **1 µF Capacitor**

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

3 Details

3.1 Hardware

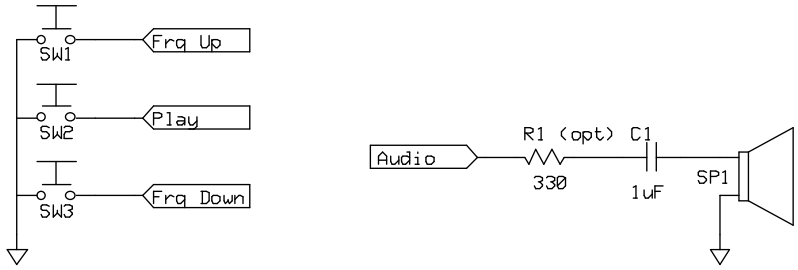

Figure 1. Schematic diagram

Use three momentary switches SW1-SW3 to control the device. Switches need full circuit from previous labs (5V through 100k resistor and the board input on one side of the switch, ground on the other side, bridge the connection when pressing the button).

- Each switch should have its own “dedicated” resistor– trying to re-use the same resistor for different buttons will end up causing undesirable connections in your circuit and strange behavior for the buzzer ultimately.

Drive a speaker SP1 from a GPIO output (labelled Audio) using capacitor C1 to block DC current. Resistor R1 is optional (but strongly recommend) and reduces the volume of the sound.

Please see the included Nucleo-L552ZE-Q pins legend (NUCLEO_L552ZE_pins.docx) for the pinout of the Arduino-included Zio connectors for:

Signal /Connection Name	Description	Direction	MCU
P_SW_UP	Period Up (SW1 from above)	Input to MCU	PA_2
P_SW_CR	Play (SW2 from above)	Input to MCU	PC_3
P_SW_DN	Period Down (SW3 from above)	Input to MCU	PB_0
P_SPEAKER	Output to Speaker (Audio from above)	Output from MCU	PD_14

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

3.2 Software

Your code in C does the following:

- The tone should sound while SW2 is pressed down.
- The period of the tone should rise while SW1 is pressed.
- The period of the tone should fall while SW3 is pressed.

The following software design was implemented:

- An initialization function which configures GPIO inputs and outputs based on which pins to which you've wired your switches and speaker.
- The delay-loop function delay_us to cycle at the correct frequency.
- A function play_tone(unsigned int duration_ms) which generates a square wave with the given period (specified in microseconds) and duration (milliseconds). This was done by toggling the audio output pin, waiting for a time delay, and repeating this process. We calculate the necessary value to pass to delay_us based on period (inverse of frequency) and the number of times to toggle the output based on period and duration.
- A function update_buttons(void) that repeatedly checks to see if any switches are pressed and responds accordingly.
    - If SW1 or SW3 is pressed, adjusts the period accordingly. Limits the value of period to within 100 microseconds and 10,000 microseconds.
    - The processor executes this loop very quickly if SW2 is not pressed, so the value of period will quickly reach the upper or lower limit if SW1 or SW3 is pressed. To slow the code down, it remembers the previous state of the buttons and only responds if the button state has changed.
- If SW2 is pressed, calls play_tone(unsigned int duration_ms).

3.3 Tasks

At each applicable step, make sure you capture pictures of the waveform and appropriate data about the waveform. Include those pictures, additional details about the waveforms as needed, and your final modified code (commenting out any code that you no longer need instead of deleting it) as your submissions

Complete these first:

1. Press the DN (down) button once to initialize the period. Play the tone by pressing the button one single time. See the output wave (aka the P_SPEAKER signal coming directly out of your board before going through the resistor and capacitor) on the oscilloscope. Verify the shape, period, and duration of these signals.
    a. Period: 374us
    b. Shape: Square wave
    c. Duration: The segment of the signal shown in figure 1 is approximately 1ms but should play for a total of 50ms.

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

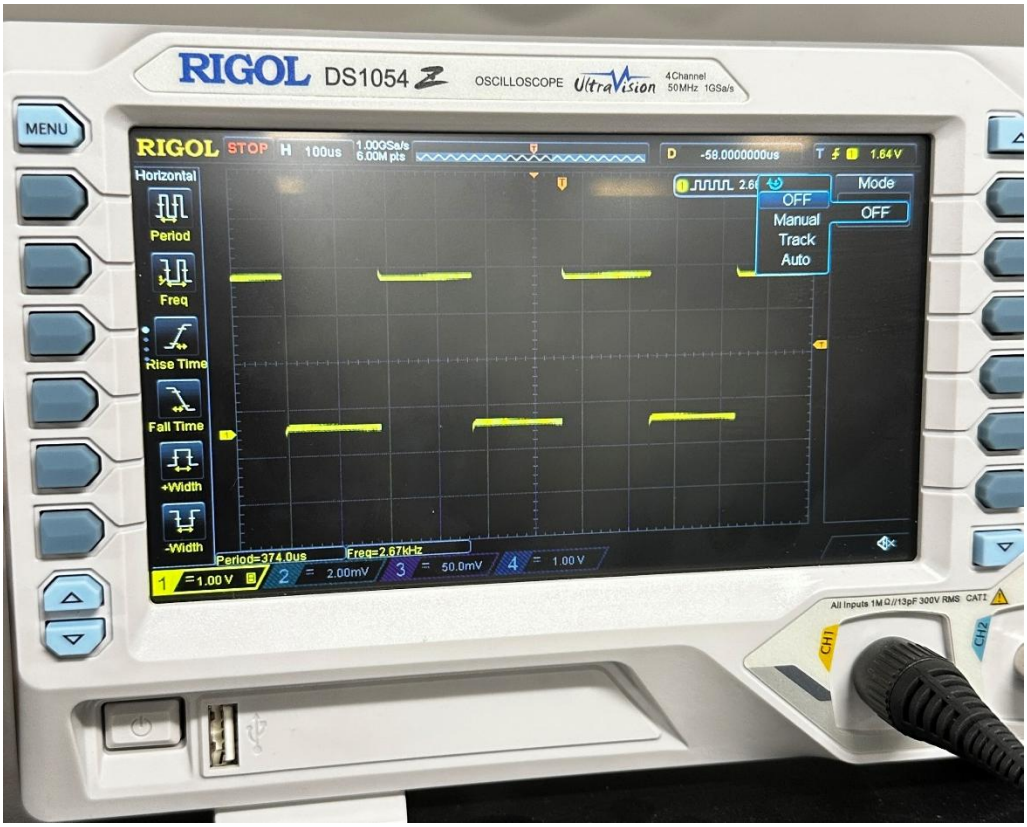

Figure 1. Output wave after pressing down once

IMPORTANT NOTE: The period you see on the oscilloscope for one full output cycle (the length of time where the output is low plus the length of time where the output is high) should be roughly 3 times larger than the period noted in the code. This is normal due to the way we are handling delays in the code – don't worry about it!

The assembly "chunk" in the code:

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

```
__asm(
"MOV       r0,#0xc0 \n\t\
MOVT      r0,#0x2000 \n\t\
LDR       r1,[r0,#0] \n\t\
ADDS      r1,r1,#0x64 \n\t\
STR       r1,[r0,#0] \n\t\
LDR       r0,[r0,#0] \n\t\
MOV       r1,#0x2711 \n\t\
CMP       r0,r1 \n\t\
BLT       END \n\t\
MOV       r1,#0xc0 \n\t\
MOVT      r1,#0x2000 \n\t\
MOV       r0,#0x2710 \n\t\
STR       r0,[r1,#0] \n\t\
"
"END: \n\t");
```

is the replacement for this commented out portion of c code:

```
//     period += 100;
//     if (period > 10000) {
//       period = 10000;
//     }
```

2. Describe what it is doing at the assembly level. Look up any instructions that you are unfamiliar with. Modify the assembly code such that the upper limit of the period is 1000 instead of 10000.
    a. For the first two lines, we can't initialize a 4 digit hex so at first we set r0 to c0 and then use mov top to make r0 20c0. Then we load that into r1 with an offset of 0x64 or 100 in decimal. It gets stored and then we move 1000 with an offset of 1 to r1 as the period. The 0x64 is for increasing or decreasing. After that it compares to make sure the period hasn't exceeded 10000 and then it allows to use the add function from before to increase by 100. Finally the period gets stored. By changing 2710 to c3e8, we make the highest period 1000, rather than 1000 and including the offset of 1 makes it c3e9.
    b. Modified code:

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

```
//     __asm(
//     "MOV      r0,#0xc0 \n\t\
//     MOVT     r0,#0x2000 \n\t\
//     LDR      r1,[r0,#0] \n\t\
//     ADDS     r1,r1,#0x64 \n\t\
//     STR      r1,[r0,#0] \n\t\
//     LDR      r0,[r0,#0] \n\t\
//     MOV      r1,#0x3E9 \n\t\
//     CMP      r0,r1 \n\t\
//     BLT      END \n\t\
//     MOV      r1,#0xc0 \n\t\
//     MOVT     r1,#0x2000 \n\t\
//     MOV      r0,#0x3E8 \n\t\
//     STR      r0,[r1,#0] \n\t\
//     "
//     "END: \n\t");
```

Figure 2. Modified assembly code with a period of 1000

Then complete these to experiment with your new version of the code:

3. Press the DN (down) button once to initialize the period. Play the tone by pressing the button one single time. See the output wave on oscilloscope. Verify the minimum period.
    a. Minimum period: 375us

Figure 3. Output signal after pressing the down button once

4. Increase the period multiple times (by pressing the UP button) until you reach the maximum. Play the tone by pressing the button one single time. See the output wave on oscilloscope. Verify the maximum period.
    a. Maximum period: 3.080ms

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

Figure 4. Output signal after pressing up the maximum amount of times

Comment out the assembly chunk and bring back in the C code before moving on!
In the C code you bring back in, change the 10000 to 1000 to replicate your change in the assembly code.

```
    duration_ms += 100;
    if (duration_ms > 1000) {
      duration_ms = 1000;
    }
  }
```

Figure 5. C code changed to have a 1000 period

Then make some modifications to the program to accomplish the following:

5. Modify the functions of SW1 (P_SW_UP in the code) and SW3 (P_SW_DN in the code) from adjusting the “period” to adjusting the “duration_ms”.

As part of this, you should:

a. Create a variable called duration_ms very similar to your period variable near the top of the code

```
7  static int duration_ms;
```

Figure 6. New variable “duration_ms”

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

b. Replace the “50” that we are currently always using as the duration to play the sound (in the “play_tone” command near the bottom of your code)

```
int main(void) {
  switches_init();
  gpio_set_mode(P_SPEAKER, Output);

  while (1) {
    // Play if the centre button is pressed.
    if (switch_get(P_SW_CR)) {
      // Play for ~50 ms.
      play_tone(duration_ms);
    }
    update_buttons();
  }
}
```

Figure 7. Replaced 50ms

6. Modify the waveform from a square wave (with identical durations for both the high and low parts of the signal) to a rectangle wave (where the signal is high for more than 50% of the period and low for less than 50% of the period, or vice versa. The total should still sum up to 100%!). This should be done inside the for loop of the play_tone function. You may choose how to “split up” the full period between times when the signal is low vs. times when the signal is high.

- I would recommend either using modulus division like we did for the random number generation to check whether we are currently on an odd or even number of “i” in the for loop, with different behavior in each case by changing the fraction used in the “delay_us(period / 2)” line, or…
- Only go through the loop half as many times, but toggle twice per loop with delays of different lengths by duplicating and modifying the “delay_us(period / 2)” line

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

```
51
52 void play_tone(int duration_ms) {
53   unsigned int i;
54   unsigned int cycles = 2 * (duration_ms * 1000) / period;
55   for (i = 0; i < cycles; i++) {
56     gpio_toggle(P_SPEAKER);
57     if (i % 2 == 0){
58     delay_us(period / 1);
59     }
60     else{
61       delay_us(period / 3);
62     }
63   }
64 }
```

Figure 8. Code to make low/high signals longer than the other

Figure 9. Output as a result of the code where the low signal is longer than the high signal.

Continue on to the next page!

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

At the top of your code where you declare the variable names, initialize the “period” variable to a value of your choice between the original default upper and lower bounds visible in the original code (I would recommend on the lower end of the range). Then complete these to experiment with your new version of the code:

```
6  static int period = 140;
7  static int duration_ms;
```

Figure 10. New variable period with a value of 140

7. Press the DN (down) button once to initialize the duration. Play the tone by pressing the button one single time. See the output wave on oscilloscope. Verify the minimum duration.
Minimum duration: 920.0ms

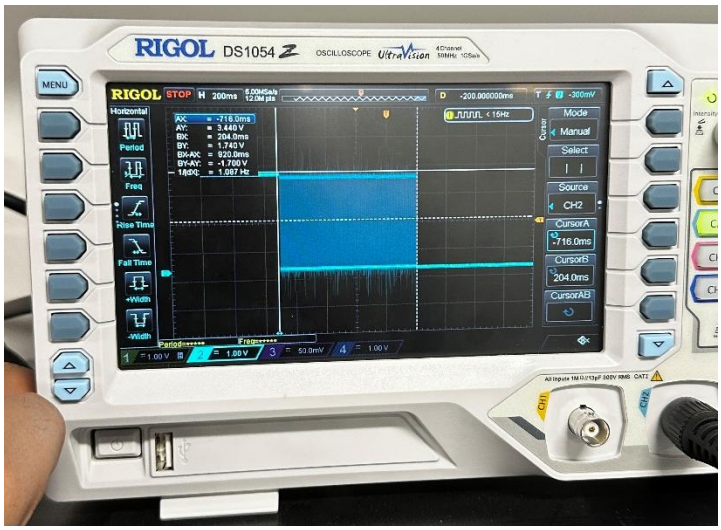

Figure 11. Minimum duration of the tone

8. Increase the “duration_ms” multiple times (until you reach the maximum!). Play the tone by pressing the button one single time. See the output wave on oscilloscope. Verify the maximum duration.
Maximum duration: 4.580s

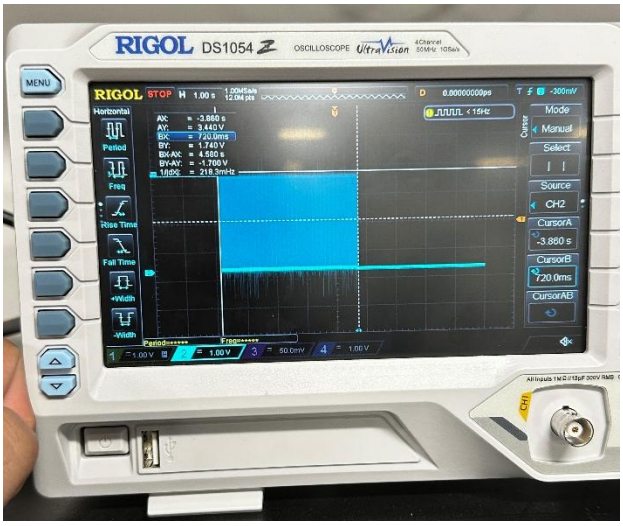

Figure 12. Maximum duration of the tone

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.

As an example, here is a waveform captured on the oscilloscope with the minimum duration to showcase capturing the period and having a rectangular wave (though you should also zoom waaayyyy out in the horizontal/time direction so that you can capture the full duration of the signal):
.\\\

Copyright © 2025 Arm Limited (or its affiliates). All rights reserved.